Quantification Under Class-Conditional Dataset Shift

David James Frederick Spence

A thesis presented for the degree of

Doctor of Philosophy

October 2018

Declaration

- This thesis, in full or in part, has not been previously submitted to this or any other university for a degree.
- None of the material in this thesis has already been submitted as part of required coursework at any university.
- No part of the thesis results from joint work with other persons other than the development of the 4dSDA dataset as set out in Section 3.4.1.

_____

David James Frederick Spence

Date: _____

Abstract

Classification is the estimation of the class of *each instance* in a dataset, *quantification* is the estimation of the *number* of instances of each class in a dataset. Quantification methods typically assume that the data which is being quantified has the same class-conditional distribution as the data on which the quantifier was trained. This thesis addresses the situation where this assumption cannot be made, where there is class-conditional dataset shift between the training data and the test data. The work was motivated by sentiment analysis tasks using tweets on Twitter. By selecting users based on the content of their tweet, the users cannot be considered to have been randomly drawn from the population. In this thesis, domain adaptation methods from classification have been applied to the problem of quantification. Separating the data into explicit sub-domains and quantifying each sub-domain separately can increase quantification accuracy but under certain conditions it can also decrease it. An expression for expected quantification error was derived in closed-form with some simplifying assumptions. In tests on real datasets, a method based on this approach gave a modest improvement to quantification accuracy. Constructing a new feature representation has proved successful for domain adaptation in classification. An approach using Stacked Denoising Autoencoders to generate a new feature representation gave a 3.3% relative improvement in quantification accuracy. Finally, a method based on using Kernel Mean Matching for weighting instances in the training set gave a relative improvement in quantification accuracy of 10.7%. Experiments were conducted on publicly available datasets and also on a custom dataset of Twitter users.

Acknowledgements

I would like to thank...

David Weir and Novi Quadrianto for supervising me.

Luc Berthouze for chairing my thesis committee.

My fellow members of the department, in particular Chris Inskip, Oliver Thomas, Matti Lyra and Miro Batchkarov for helping me get things to work.

My friends Ian Handel and Mark Bronsvoort at Edinburgh University and Neil Hawkins at Glasgow University for their general advice on doing a PhD and their specific advice, particularly on statistics.

Thomas Kober for gender labelling the 4dSDA dataset.

Chris Inskip for the use of the SDA dataset.

CASM Consulting LLP for the use of Method52.

Katie Barnett for putting up with me.

Contents

List of Figures

List of Tables

Variables

Style	Example	Use
Italic Roman lowercase	n_t	Scalars
Bold Roman lowercase	$\mathbf{a}_{vs}$	Vectors
Italic Roman uppercase	P	Random Variables
Roman uppercase	R^*	Matrices
Bold Italic Roman uppercase	$\boldsymbol{P}$	Vector of Random Variables

$\mathbf{a}_v$	Vector of actual counts by class (main-class and sub-domain) in the validation dataset
$\mathbf{a}_t$	Vector of actual counts by class (main-class and sub-domain) in the test dataset
$\mathbf{a}_{nt}$	Vector of actual counts by class (main-class only) in the test dataset
R^*	Matrix of actual-to-predicted class probabilities implicit in the classifier
R_v	Matrix of actual-to-predicted class (main-class and sub-domain) ratios as observed for the validation dataset
$\mathbf{p}_v$	Vector of predicted counts by class (main-class and sub-domain) as generated by the classifier from the validation dataset
R_{nv}	Matrix of actual-to-predicted class (main-class only) ratios as observed for the validation dataset
$\mathbf{p}_t$	Vector of predicted counts by class (main-class and sub-domain) as generated by the classifier from the test dataset
$\mathbf{p}_{nt}$	Vector of predicted counts by class (main-class only) as generated by the classifier from the test dataset
$\hat{\mathbf{a}}_t$	Vector of estimated actual counts by class (main-class and sub-domain) for test dataset using sub-domain method
$\hat{\mathbf{a}}_{tm}$	Vector of estimated actual counts by class (main-class only) in the test dataset using sub-domain method
$\hat{\mathbf{a}}_{nt}$	Vector of estimated actual counts by class (main-class only) in the test dataset using no-sub-domains method
$\mathbf{e}_{sd}$	Error between estimated and actual class counts in the test dataset from method using sub-domains
$\mathbf{e}_{nsd}$	Error between estimated and actual class counts in the test dataset from method not using sub-domains
n_v	The size of the validation dataset
n_t	The size of the test dataset

Chapter 1

Introduction

How do you accurately estimate the class proportions in a dataset when the class-conditional feature distribution is different to that of the dataset that is available for training?

Often it is not the class of *individual* data points that is of interest, it is the *distribution* of classes in the whole dataset. In sentiment analysis it might be the proportion of a group that is positive about a product. In market research it might be the ratio of men to women in a group of respondents and in epidemiology it might be the prevalence of a disease in a population. Increasingly, the requirements for privacy and anonymity mean that analysis has to be presented as group aggregates with any information at individual level removed. Estimating the class distribution in a dataset instead of the classes of individual data points has been termed *quantification* [42].

The naïve approach (*classify and count*) is simply to classify the data with a classifier and count the number of instances assigned to each class. However this approach is flawed because classifiers are imperfect. Consider Figure 1.1. Given a test sample that is made up of 100% positive instances, a classifier will only classify some of those instances as positives. The proportion of actual positive instances that are classified as being positive is the True Positive Rate (tpr). Similarly when the test sample is made up of 100% negative instances, a classifier will inevitably classify some of those instances as positives. The proportion of actual negative instances that are classified as positive is the False Positive Rate (fpr).

For some value of class distribution m^* the number of false negatives will exactly offset the number of false positives and the estimated class distribution will be correct. For all

other class distributions the estimate will be incorrect.

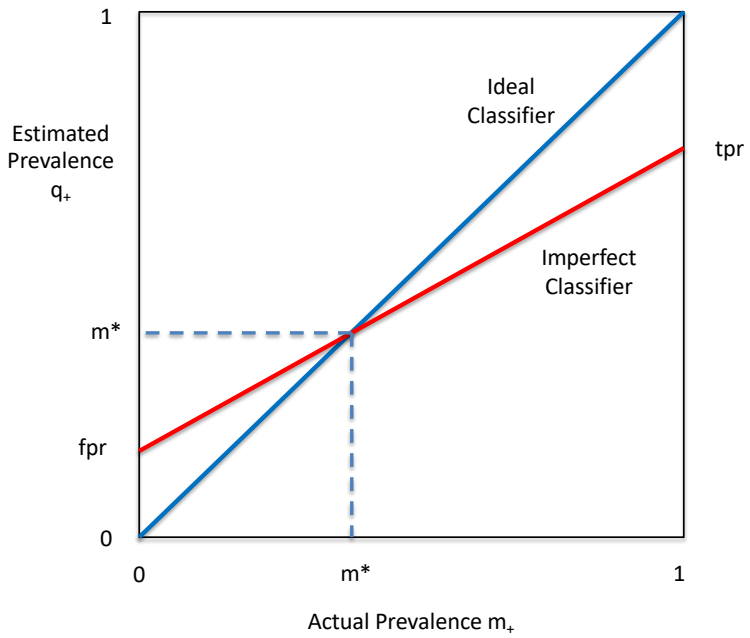

Figure 1.1: Estimated and actual class distribution with the *classify and count* method

This can be relatively simple to correct. *Classify and adjust* methods apply an adjustment to the output of the classifier that corrects for its imperfect classification. Figure 1.2 shows the quantification performance of a *classify and count* method and a *classify and adjust* method. The estimates from the *classify and count* method are as would be expected from Figure 1.1 while the improvement in estimation accuracy from the *classify and adjust* method is clear.

Classify and adjust methods typically require the assumption that the performance of the classifier on each class in the test data is the same as it was on each class in the original labelled training data. This implies that the class-conditional feature distribution in the test data, $P_{te}(x|y)$, is the same is it is in the training data, $P_{tr}(x|y)$. As per the usual convention, x represents the *features* of the data while y represents the *class labels*.

However, the class-conditional feature distribution may *not* the same in both the training and test sets, for example:

A 'quantifier' has been trained to give an estimate of the male/female gender balance in a group of individual Twitter users based on the accounts that they are following on Twitter. The quantifier has been trained with a broad set of UK Twitter users with each user correctly labelled as male or female. The quantifier is then used to estimate the gender

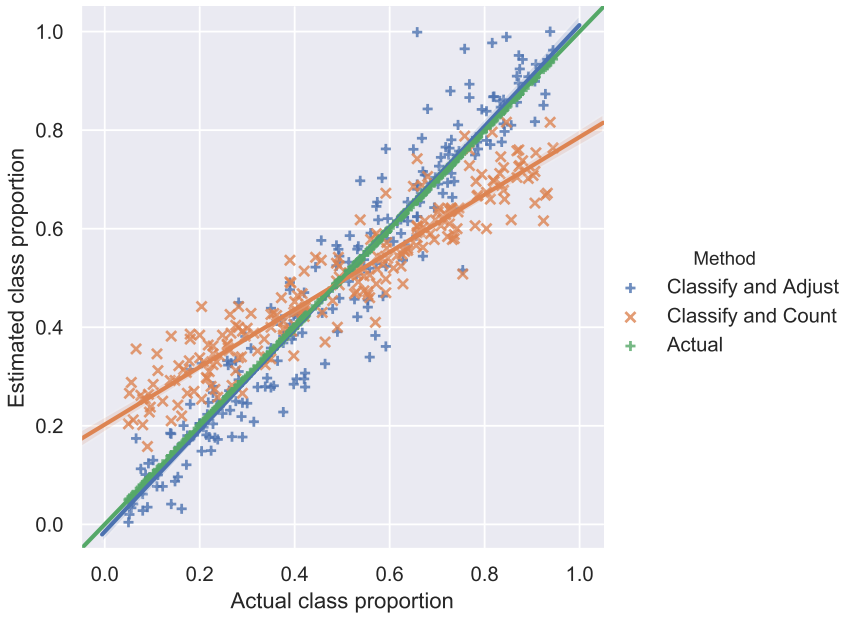

Figure 1.2: Estimated vs. actual class proportions using both the *classify and count* and the *classify and adjust* methods. UCI_dev datasets. Data from Section 5.9

balance of another group of individuals. This group of individuals have all been selected because they tweeted about retirement homes in Scotland.

This group of Twitter users, selected from a population on the basis of the content of what they say in a tweet, are unlikely to have the same class-conditional feature distribution as the group of users on which the quantifier was trained. In this example, within each class (male and female) the age and location distribution of the dataset in question is likely to be different. This difference *could* result in an error in the estimation of the gender balance of the group. In general, we *cannot* simply assume that a dataset that has been generated as a result of some selection process has the same class-conditional feature distribution as the dataset on which the quantifier was originally trained.

When class-conditional feature distributions are different we refer to this as class-conditional dataset shift. Figure 1.3 shows how quantification accuracy with a standard *classify and adjust* method degrades with increasing class-conditional dataset shift[1].

The majority[2] of academic work on quantification makes the assumption that class-

[1] see Section 5.1 for an explanation of PADcb

[2] e.g. [11] [13] [35] [37] [42] [49] [68] [97] [117] [121]

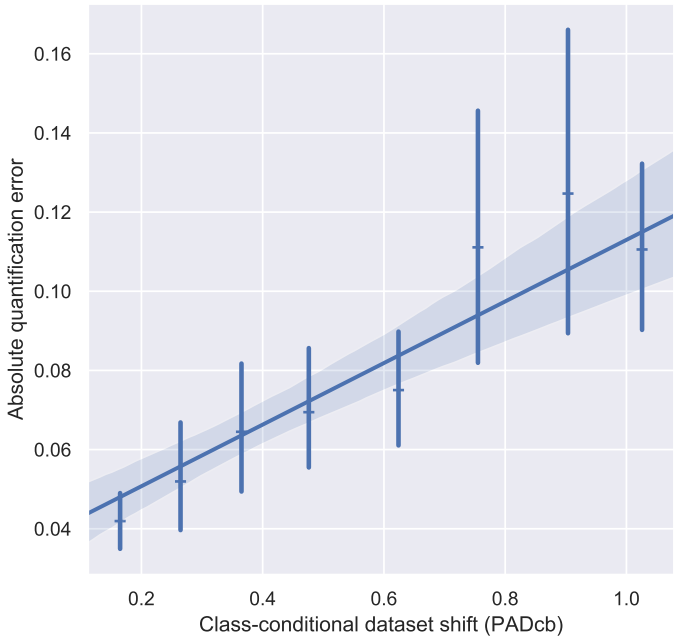

Figure 1.3: Absolute quantification error using classify and adjust method vs. class-conditional dataset shift. Data from Section 5.9. 95% confidence intervals.

conditional dataset shift has *not* occurred. The novelty in this thesis is that its focus is on quantification when class-conditional dataset shift *has* occurred. Its main contribution is to demonstrate that several approaches can reduce quantification error under conditions of class-conditional dataset shift. The best of these, using importance weighting to select from the labelled validation data, gave a relative improvement in quantification error of 10.7% over the *classify and adjust* baseline on datasets where the class-conditional feature distribution is different from that of the training data.

In this thesis, three different approaches are applied to the problem of *quantification under class-conditional dataset shift*: *explicit sub-domains* in Chapters 3 and 4; *importance weighting of instances* in Chapter 5 and *feature representations* in Chapter 6. However, all of these methods fundamentally address the problem in a similar way: applying a *domain adaptation* step to reduce (or ideally eliminate) the class-conditional feature distribution difference between the training and test sets so that a standard quantification method that relies on the assumption of no change in class-conditional feature distribution can work effectively. This is shown diagrammatically in Figure 1.4.

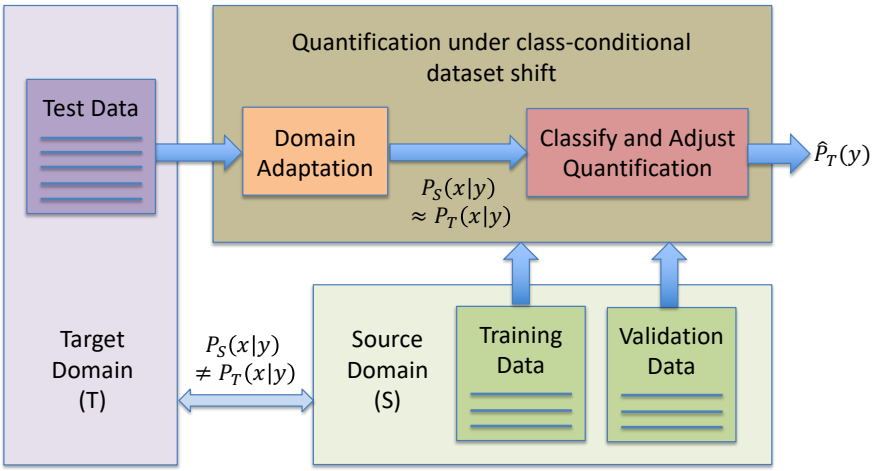

Figure 1.4: Common approach to quantification under class-conditional dataset shift

The *explicit sub-domains* approach in Chapters 3 and 4 can be thought of as a 'divide and conquer' approach. In these chapters the assumption is made that the data can be broken down into smaller groups ('sub-domains') in which the conditional feature distributions *do not* vary. Quantification is carried out at this sub-domain level and the results then aggregated up to class level as a final step. A limitation of this method is that the sub-domain has to be identified up-front and labelled in the training data. Another drawback of this approach is that dividing the training set into smaller groups increases the relative level of 'noise'. The question explored in these chapters is whether the advantages of quantifying in smaller sub-domains outweighs the disadvantages from increased noise. An analytic approach yielded a closed-form answer to this question but only when the assumption was made that there is a large amount of labelled data available at training time. Numerical simulation allowed the question to be explored without making simplifying assumptions and showed the significance of a number of parameters.

In Chapter 4, the insights from Chapter 3 are used to develop a method which utilises sub-domains only when doing so was likely to improve quantification accuracy. This method achieved a 4.5% relative improvement in mean absolute error over the baseline method on the test data.

In Chapter 5, *importance weighting of instances* methods are used for *domain adaptation*. With these methods, the distribution of the training data is brought closer to the distribution of the test data by applying 'importance' weights to individual instances in the training set. Several methods of computing importance weights are used including Sample Selection Bias Correction [122], Kernel Mean Matching [58] and Unconstrained

Least-Squares Importance Fitting [104]. These methods work on aligning data distributions *overall*, not specifically the *class-conditional* distribution differences, so various approaches were taken to address the issue of class-conditionality. Ultimately, a method using Kernel Mean Matching gave a 10.7% relative improvement in mean absolute error over the baseline method on the test data.

In Chapter 6, attention switches from *instances* to *features*. *Domain adaptation* is addressed by using the Marginalised Stacked Denoising Autoencoder (mSDA) method [29] to transform the original features into a new representation. The best paramater settings gave a 3.3% relative improvement in mean absolute error over the baseline method on the test data.

The relevant literature is reviewed in Chapter 2 and conclusions and directions for potential further work are set out in Chapter 7.

The motivation for this work came from the Polly project in 2014-15. The Polly project was a collaboration between the University of Sussex, CASM Consulting LLP, the market research firm Ipsos-Mori and the think-tank Demos. It was funded by the UK Technology Strategy Board (now Innovate UK), the EPSRC[3] and the ESRC[4]. A key part of the project was to explore the potential for the demographic profiling of Twitter users and promising results were obtained for estimating age, gender and location. The question of a dataset shift between the dataset being tested and the dataset on which the classifiers were trained was acknowledged as a possible issue but was not explicitly addressed as part of the project.

[3]The Engineering and Physical Sciences Research Council

[4]The Economic and Social Research Council

Chapter 2

Literature review

This Chapter is organised into four sections. Literature on *quantification* is explored in Section 2.1, *dataset shift* in Section 2.2 and *unsupervised domain adaptation* in Section 2.3. Finally, literature that has brought together *domain adaptation* and *quantification* is reviewed in Section 2.4.

Throughout the thesis, data that is used for training purposes is referred to as the *training set* and we say that it has been drawn independently and identically distributed (iid) from the *Source* domain. Similarly, we say that the data that is used for testing purposes is the *test set* which has been drawn iid from the *Target* domain.

2.1 Quantification

The approaches to quantification can be grouped into four broad categories:

- Classify and count
- Classify and adjust
- Distribution matching
- Direct quantification

2.1.1 Classify and count

For the reasons outline in the Introduction, *classify and count* is a naïve approach to quantification. The inevitable imperfection of the classifier gives an inevitably imperfect method of estimating class proportions.

2.1.2 Classify and adjust methods

Classify and adjust methods still rely on a trained classifier to classify or assign class probabilities to the data but then a second *adjustment* step is applied to arrive at the actual estimate of class proportions. The adjustment step implicitly or explicitly relies on information about the relationship between the actual and estimated class labels that has been obtained from labelled data, typically at the time that the classifier is trained. The two most common approaches to making this adjustment are matrix-inversion and probabilistic expectation-maximisation.

2.1.2.1 Matrix-inversion

The error from the *classify and count* method can be corrected with a simple linear transformation. Assume we have two classes, positive (+) and negative (-). The test set contains n_t instances of which a_+ are in the positive class. We define m_+ as the proportion of actual positive class instances in the test set:

$$m_+ = \frac{a_+}{n_t}, \tag{2.1}$$

and $\hat{m}_+$ as the estimate of this value. Counting the instances by predicted class as given by the classifier (i.e. the *classify and count* method) we have p_+ positives. We define q_+ as the proportion of predicted class positive instances in the test set where:

$$q_+ = \frac{p_+}{n_t}. \tag{2.2}$$

The *matrix-inversion* estimate of m_+ is then [42]:

$$\hat{m}_+ = \frac{q_+ - fpr}{tpr - fpr}, \tag{2.3}$$

where the True Positive Rate (tpr) and False Positive Rate (fpr) are calculated from labelled *validation* data normally at the same time that the classifier is trained but which

has not been used to train the classifier. This method relies on the assumption that the computed *tpr* and *fpr* values are the *same* as would be seen on the test data if its labels were available for inspection i.e. that the class-conditional feature distribution in the Target domain is the *same* as that in the Source domain.

In the field of machine learning, the matrix-inversion method is usually credited to Forman [42] as the *Adjusted Count* method, although this is fundamentally the same method that was seen earlier in Vucetic and Obradovic [117]. In epidemiology this method has been used since at least the 1960s. Rogan and Gladen [95], Levy and Kass [81] and Buck et al. [21] are widely cited.

Forman [44] puts forward a range of variations on the *Adjusted Count* method that are aimed at prioritising quantification performance over classification performance. These are shown in Table 2.1.

Table 2.1: Variations on the *Adjusted Count* method from Forman [44]

Method	Description
Crossover	Set the threshold of the classifier such that it gives $fpr = (1 - tpr)$
T50	Select the classifier threshold that results in $tpr = 50\%$
Max	Select the classifier threshold that results in the maximisation of the denominator i.e. maximise (*tpr* - *fpr*)
Median Sweep	Compute $\hat{m}_+$ for *every* setting of the classifier threshold and return the *median* of these values

Forman [44] observes that *Median Sweep* performs better than the *Mixture Model* and *Adjusted Count* methods although he suggests though that *T50* and *Crossover* may be simpler to implement.

While many further works discussed in this section claim to have achieved higher levels of performance, the performance of the simple matrix-inversion method has proved to be strong. It has frequently been used as a baseline and has frequently been shown to perform on a similar level to the author's own chosen approach (e.g. Barranquero et al. [12] Esuli and Sebastiani [37] Gao and Sebastiani [49] Milli et al. [86] Xue and Weiss [121]). González et al. [52] states that the AC method is a theoretically perfect method when its learning assumptions are fulfilled i.e. if we have perfect estimates for *tpr* and *fpr* for the data on which the quantification is being performed.

Bella et al. [13] put forward a variation on *Adjusted Count* method which they called *Scaled Probability Average*. They believe that class-probabilities offer richer information on the dataset than estimated class labels. In this method the aggregated class probabilities from a classifier are first computed and then are adjusted in a similar manner to the way that Forman [44] adjusts the counts by class with the *Adjusted Count* method. Bella et al. [13] found that their *Scaled Probability Average* method outperformed other methods including the Forman [44] *Adjusted Count* method. However, Esuli and Sebastiani [37] performed a number of tests and found that *Scaled Probability Average* was *not* consistently better than *Adjusted Count*.

Further details of the Forman [42] *Adjusted Count* method can be found in Section A.1

2.1.2.2 Probabilistic expectation-maximisation

In 2002 Saerens et al. [97] outlined a probabilistic expectation-maximisation method for estimating class distribution that is considered to be seminal [35] and arguably the most popular algorithm for quantification [52].

The method requires a classifier that generates an output that can be interpreted as the probability of being a member of each class, $P(y|x)$. In the matrix-inversion method above, the information about the probabilistic relationship between actual and predicted class labels is *explicit* in the values of *tpr* and *fpr*. In this method this information is *implicit* in the training of the classifier and the class label probabilities that it assigns.

The classifier is trained on the labelled training data and then used to assign estimated class probabilities to the test data from the Target domain, $P_T(y|x)$. After that a two step adjustment process iterates until a convergence criteria is met:

1. The class distribution $P_T(y)$ is re-estimated by marginalising the latest estimate of posterior class probabilities $P_T(y|x)$.
2. The posterior class probabilities $P_T(y|x)$ are re-estimated using the latest estimate of the class-distribution $P_T(y)$.

Further details of the Saerens et al. [97] method can be found in Section A.2.

2.1.3 Distribution matching

The observed distribution of features, $P_T(x)$, or estimated class probabilities, $P_T(\hat{y})$, in the Target domain is assumed to be a mixture of the class-conditional feature distributions in the Source domain observed at training time.

The class distribution in the Target domain is estimated by comparing the test set to a synthetic dataset which has been made up by sampling the labelled data from the Source domain to a given class proportion. The estimate of the class proportion in the Target domain is the class proportion in the synthetic set which minimises some measure of distance between the distribution of the synthetic set and the distribution of the test set. Different authors have used different distance metrics.

Clearly these methods are again dependent on the assumption that the class-conditional feature distribution is the same in the Source and Target domains.

2.1.3.1 Distribution matching in the estimated label space $\hat{\mathbf{Y}}$

Forman [42] put forward the *Mixture Model* method. Firstly, a classifier is trained with data from the Source domain, then other labelled data from the Source domain is put through the trained classifier to give a distribution of raw classifier output values for each class.

Forman [42] uses a measure he defines as *PP-Area* to measure the distance between the test and the synthetic datasets. PP-Area is defined as the area bounded by the Cumulative Distribution Function (CDF) of the test dataset and the CDF of the synthetic dataset. Forman [42] considered PP-Area to be a better metric than the more conventional Kolmogorov-Smirnov value. He found that *Mixture Model* was very resilient to wide variations in the class distribution of the training data, but was normally outperformed by the variants on the *Adjusted Count* method (*Crossover*, *T50*, *Max*, *Median Sweep* (see Section 2.1.2.1)).

GonzáLez-Castro et al. [54] measured the distance between the two distributions using *Hellinger Distance*. They found that estimating class proportions with a method based on the predicted class labels $\hat{y}$ performed better than the method based the feature distributions x (see Section 2.1.3.2).

2.1.3.2 Distribution matching in the feature space X

Du Plessis and Sugiyama [35] showed that the Saerens et al. [97] EM algorithm can be refomulated as a mixture method which minimises Kullback-Liebler (KL) divergence. They, however, prefer Pearson (PE) divergence to KL-divergence as the measure to minimise. PE-divergence can be considered to be the squared-loss variant of KL-divergence [35]. They prefer PE-divergence largely for reasons of practical implementation but also state that it has superior convergence properties. There was some difference in performance for the five methods that they implemented when they were applied to the six datasets, but in all six cases the PE-divergence based method either equalled or was better than the Saerens et al. [97] EM method.

As discussed above, GonzáLez-Castro et al. [54] explored distribution matching by minimising the Hellinger Distance in both the estimated label space $\hat{Y}$ in the previous section, and in the feature space X. They obtained better performance when working in $\hat{Y}$ and believed that the lower performance in X was down to the issue of sparseness. The approach taken by Du Plessis and Sugiyama [35] and Iyer et al. [68] to measuring distance in X is arguably more sophisticated than the Hellinger distance approach used by GonzáLez-Castro et al. [54] and this may explain its superior performance.

2.1.3.3 Distribution matching in a transformed feature space

Iyer et al. [68] and Kawakubo et al. [76] projected the distributions into a Reproducing-Kernel Hilbert Space (RKHS) and then minimised Maximum Mean Discrepancy (MMD) (see Section 2.2.3.2). Their work builds on previous work such as Saerens et al. [97], Du Plessis and Sugiyama [35] and Zhang et al. [124]. Iyer et al. [68] initially used the PE-divergence method favoured by Du Plessis and Sugiyama [35] as a baseline, but dropped it stating that it (surprisingly) performed no better than a baseline of counting by predicted class using a classifier built on the work of Sun et al. [107].

2.1.4 Direct quantification

Given that the ultimate aim is *quantification* and not *classification* an alternative approach is to learn a *quantifier* directly and not to learn a *classifier* as an intermediate step. This is typically done by minimising a loss function based on quantification error rather than

on classification error.

Esuli and Sebastiani [37] [38], Barranquero et al. [12] and Gao and Sebastiani [49] [48] all used the SVM for Multivariate Performance Measures ($\text{SVM}^{\Delta}_{multi}$) method as put forward by Joachims [72]. This method is itself a development from Tsochantaridis et al. [111].

Classifiers typically minimise a loss function where the loss is aggregated from the losses computed for each individual instance in the training set. Joachims [72] $\text{SVM}^{\Delta}_{multi}$ method is different in that it can minimise a loss function that has been computed across a *set* of data instances where the loss *cannot* be dis-aggregated to a loss for each instance, for example from a confusion matrix. This allows a classifier to be trained to directly optimise a measure of quantification accuracy.

Esuli and Sebastiani [37] [38] use the Kullback-Leibler divergence (KLD) for their loss function. Specifically the KLD between the actual class distribution and the predicted class distribution based on the count of instances by predicted class from the classifier. They compare this method (which they term SVM(KLD)) to other baseline methods such as those of Forman [44] and Bella et al. [13] and claim that it is superior in accuracy, stability and running time.

Interestingly they comment that *tpr* and *fpr* were far from invariant across different sets and they argue that this supports the SVM(KLD) method over methods such as *Adjusted Count* that require *explicit* values for *tpr* and *fpr*. However, $\text{SVM}^{\Delta}_{multi}$ / SVM(KLD) is still optimising over the full set of training data. The assumption is that the test set and the training set are be drawn iid from the same domain. The SVM(KLD) will have learnt *some* relationship between the set of x values of the instances in the training set and the set of class labels y. While the method does not rely on *explicit* values for *tpr* and *fpr* it will be just as susceptible to the underlying changes in the relationship between x and y that cause those changes in *tpr* and *fpr*.

Barranquero et al. [12] also implemented a similar quantification approach to that used by Esuli and Sebastiani [37] i.e. one based solely on quantification loss. They found that it performed poorly. They argue that in order to generalise well a *quantifier* must still be a good *classifier*. A loss function that only focuses on quantification error (such as is the case with Esuli and Sebastiani [37]) is, in their view, unsuitable because the resulting hypothesis space contains several local optima. Like Esuli and Sebastiani [37] they also use the ($\text{SVM}^{\Delta}_{multi}$) but their loss function, the *Q-function*, is a linear combination of

quantification loss function *and* a classification loss function[1]. However, when they applied the analytical approach as advised in Demšar [31] they found no statistically significant difference between this method and 6 of the 7 Forman [44] methods that were used as benchmarks.

One possible explanation for the better performance claimed by Esuli and Sebastiani [37] is that their experiments used a large number of classes (88 in one experiment and 99 in another) whereas the Barranquero et al. [12] experiment used binary classes. As the number of classes tends towards the number of data instances (i.e. to a point where each class contains just one data instance and vice versa) the error in the estimate of class distribution (the quantification error) tends towards the classification error. By choosing such a large number of classes Esuli and Sebastiani [37] are effectively incorporating a degree of classification loss into their quantification loss based approach.

Tasche [109] looked at both the Barranquero et al. [11] and Esuli and Sebastiani [37] direct quantification methods and evaluated the method from Barranquero et al. [11] both from a theoretical perspective and experimentally. He found that the method was sensitive to mis-calibration and limited in its application.

Milli et al. [86] put forward a direct quantification method that did not make use of the Joachims [72] $\text{SVM}^{\Delta}_{multi}$. Their method used decision trees (*Quantification Trees*). They reported better performance than the Forman [44] *Adjusted Count* method, although in several cases the *Adjusted Count* method actually gave the best performance. The baselines for *Adjusted Count* used classifiers with the parameters set at their default values. Finally, the class distribution of the training sets was varied between 0.05 and 0.95. There are known issues when using unbalanced datasets to train standard classifiers [26] [69] [79] and this may have a larger negative impact on the baseline SVM classifier than on their decision tree method. However, despite these reservations, there is the possibility that their method is somehow akin to building a robust feature representation as per the methods set out in Section 2.3.3 and would potentially be an interesting area for further work.

In summary, Barranquero et al. [12] re-implemented the method from Esuli and Sebastiani [37] and found it performed poorly but found that their own method is not statistically

[1]They modelled their Q-measure on F-measures for balancing recall and precision in classification

significantly better than the methods from Forman [44]. Tasche [109] also found that their method was no better than simple classify and adjust methods such as those from Forman [44]. Finally, the method in Milli et al. [86] was also not found to be clearly superior to the methods from Forman [44].

My conclusion was that none of the direct quantification methods are demonstrably superior than the far simpler *Adjusted Count* method from Forman [44] so this is the method chosen for use in this thesis.

2.1.5 Quantification loss functions

The distribution matching methods in Section 2.1.3 and the direct quantification methods in Section 2.1.4 minimise a loss function over a set of instances rather than aggregate a loss function calculated separately for each instance in a set. Different authors have used a variety of loss functions, some of which are listed in Table 2.2 below:

Table 2.2: Quantification loss functions

	Title	Used in	Note
NSS	Normalised Square Score	[12]	
NAS	Normalised Absolute Score	[12]	
EMD	Earth Mover's Distance	[62] [76]	see 2.2.3.3
HDx	Hellinger Distance in X	[54]	
HDy	Hellinger Distance in y	[54]	
PE	Pearson Divergence	[35]	
MMD	Maximum Mean Discrepancy	[68] [76]	see 2.2.3.2
PP-Area	PP-Area	[44]	

2.1.6 Quantification test methods

The normal method (e.g. Forman [44], Bella et al. [13], Barranquero et al. [12]) for testing for quantification accuracy is to construct test datasets of a given class distribution by separately sampling instances of each class from the available labelled data. This is the approach I have used.

However, Esuli and Sebastiani [37] prefer to use the datasets in their 'natural' state i.e.

without any adjustment to the class proportions. They argue that artificially adjusting class proportions would create unrealistic datasets. González et al. [52] are sympathetic to this approach, observing that the class-conditional sampling methods may be artificial with respect to the actual data distribution of the problem.

This is an interesting question for further work and is discussed further in Chapter 7.

2.1.7 Quantification applied to specific areas

In many papers the motivation to explore quantification is driven by a particular problem in a particular field.

In epidemiology, estimating disease prevalence using screening tests is effectively the equivalent of *quantification* in computer science [87]. The limitations of the simple *classify and count* method with an imperfect test are well known and the matrix-inversion formula that computer scientists credit to Forman [42] in 2005 has been used by epidemiologists for estimating disease prevalence since Rogan and Gladen [95] in 1978, Levy and Kass [81] in 1970 or Buck et al. [21] in 1966.

However, while epidemiologists have been working with the problem of quantification with imperfect classifiers for many years (e.g. Cowling et al. [30], Donald et al. [33], Greiner and Gardner [56], Joseph et al. [73], McV Messam et al. [85]) they do not appear to have developed methods for dealing with quantification under dataset shift that could be used as part of the work for this thesis.

In social science, Hopkins and King [65] point out that practitioners want generalisations about the population of documents rather than the classification of individual documents i.e. quantification rather than classification. The focus of their work is on quantifying electronic records (blogs, speeches, government records, newspapers etc.) by category.

Finally, sentiment analysis is an area which features heavily in the works on quantification. Works in this area include Blitzer et al. [19], Chan and Ng [25], Esuli et al. [38], Amati et al. [6], Chan and Ng [24] and Gao and Sebastiani [48]. By its nature, users of sentiment analysis tend to be interested in the aggregate opinion of a group rather than the opinion of individuals.

2.2 Dataset shift

Dataset shift is when the joint distribution between features x and labels y in the Target domain T differs from that in the Source domain S [93] i.e.

$$P_T(x, y) \neq P_S(x, y) \tag{2.4}$$

Bayes' rule gives:

$$P(x, y) = P(x|y)P(y) = P(y|x)P(x). \tag{2.5}$$

Using Bayes' rule, Moreno-Torres et al. [88] put forward the taxonomy for types of dataset shift shown in Table 2.3.

Table 2.3: Types of dataset shift [88]

		Conditional	Marginal
Prior probability shift:		$P_T(x\|y) = P_S(x\|y)$	$P_T(y) \neq P_S(y)$
Covariate shift:		$P_T(y\|x) = P_S(y\|x)$	$P_T(x) \neq P_S(x)$
Concept shift:		$P_T(x\|y) \neq P_T(x\|y)$	$P_T(y) = P_S(y)$
	or	$P_T(y\|x) \neq P_T(y\|x)$	$P_T(x) = P_S(x)$
Other shift:		$P_T(x\|y) \neq P_T(x\|y)$	$P_T(y) \neq P_S(y)$
	or	$P_T(y\|x) \neq P_T(y\|x)$	$P_T(x) \neq P_S(x)$

Most works on quantification assume *Prior Probability Shift* [68] i.e. that while the class distribution is different in the Target domain to the Source domain $P_T(y) \neq P_S(y)$, the class-conditional feature distributions are not i.e. $P_T(x|y) = P_S(x|y)$.

In this thesis, the assumption is that the class-conditional feature distribution $P(x|y)$ is *not* the same in both the Source and Target domains. Assuming both that $P_S(y) \neq P_T(y)$ and $P_S(x|y) \neq P_T(x|y)$ would be classified as *other dataset shift* in Moreno-Torres et al. [88]. They regard these problems are so hard that they are currently 'impossible' to solve. However, several works have attempted to address this 'impossible' problem, typically by applying some form of constraint between the Source and Target domain.

Throughout this thesis the term *bias* is used interchangeably with the term *dataset shift*.

2.2.1 Causality and dataset shift

Both Moreno-Torres et al. [88] and Storkey [103] link type of dataset shift to the direction of causality. A common point of reference for both is Fawcett and Flach [39], which is itself a response to Webb and Ting [118]. Moreno-Torres et al. [88] states that writing the joint distribution as $P(x|y)P(y)$ applies only to $Y \rightarrow X$ problems. However González et al. [53] think that this is not correct. Their view is that the property that $P(x|y)$ remains unaltered must be analysed for each particular application, independently of whether it belongs to $X \rightarrow Y$ or $Y \rightarrow X$ problems.

2.2.2 Causes of dataset shift

Storkey [103] considered the reasons for dataset shift and proposed the six categories given in Table 2.4 below:

Table 2.4: Reasons for *dataset shift* from Storkey [103]

Simple Covariate Shift	Only the distributions of X change, everything else stays the same
Prior Probability Shift	Only the distribution of Y changes, everything else stays the same
Sample Selection Bias	Distributions differ as a result of an unknown sample rejection process
Imbalanced Data	Deliberate dataset shift for computational or modelling convenience
Domain Shift	Changes in measurement
Source Component Shift	Changes in strength of contributing components

While these are given as causes for dataset shift, they are in reality a mix of causes and types. Moreno-Torres et al. [88] makes a clearer separation of causes and types. They state that while there are a variety of potential causes the most important causes of dataset shift are *sample selection bias* and *non-stationary environments*.

Sample selection bias is itself a major area of study. With over 27,000 citations Heckman [60] is regarded as the seminal work on the subject and is the field of study for which he won the Nobel prize in Economic Science. Zadrozny [122] applied Heckman's methods

for correcting for *sample selection bias* to the world of machine learning and these are discussed in Section 2.3.2.1.

The concept of *sample selection bias* is very relevant to this thesis. Going back to the motivating in Chapter 1, *sample selection bias* has occurred because the group of individuals that we are looking to quantify have been selected on the basis of the content of their tweets. They are not a simple random iid sample of Twitter users.

By *non-stationary environments*, Moreno-Torres et al. [88] are considering environments where the data is non-stationary in *time* or non-stationary in *space*. They give the example of junk mail as an example of a non-stationary environment in time. The creators of junk mail change the content and format of the junk mails they generate to attempt to defeat advances in junk mail filtering. Kelly et al. [77], Gama et al. [45] and others address dataset shift as temporally non-stationary environment problem and in this area the term *drift* is typically used.

Sample selection bias and *non-stationary environments* can be seen as two ways of looking at the same issue: *non-stationary environments* can be considered as the equivalent of *sample selection bias* where the data in the Target domain has been sampled with *sample selection bias* biased on either time or on space.

2.2.3 Measures of dataset shift

Measuring the shift between datasets is not a trivial problem. The three main approaches appear to be:

- $\mathcal{A}$-distance
- Maximum Mean Discrepancy
- Earth Mover Distance

2.2.3.1 $\mathcal{A}$-distance

$\mathcal{A}$-distance was originally defined in Kifer et al. [78] where they state that the intuitive meaning of $\mathcal{A}$-distance is that it is the largest change in probability of a set that the user cares about. The authors were looking for a distance function that would detect a distance $> \epsilon$ between two distributions P_1 and P_2 with a sample of at most n points from each

of P_1 and P_2. They considered, and rejected several existing measures. They rejected Jensen-Shannon Divergence because it can only be applied to discrete distributions and because they felt that the concepts of entropy on which it is built are hard to convey to end users. They rejected other common measures of distance between distributions because they are too sensitive (e.g. L_1) or too insensitive (e.g. L_p with $p > 1$)[2].

However, the authors state that in practice, computing the exact $\mathcal{A}$-distance is impossible and that one has to compute a proxy. Ben-David et al. [14] showed that a proxy for $\mathcal{A}$-distance can be found by optimising a classifier to discriminate between the two datasets and observing the error.

Proxy $\mathcal{A}$-distance $\hat{d}_{\mathcal{A}}$ is defined as:

$$\hat{d}_{\mathcal{A}} = 2(1 - 2\epsilon), \tag{2.6}$$

where ϵ is the error rate obtained with the best hypothesis from the hypothesis set.

This has an intuitive meaning: if an optimised classifier cannot distinguish between two equal-sized datasets then they are close. In this case the classifier error rate will be around 0.5 giving a $\hat{d}_{\mathcal{A}}$ of around 0.

Using classification accuracy as a measure of dataset shift was also used by Torralba and Efros [110] to measure the similarity between image datasets. Similarly the sample selection bias correction method from Zadrozny [122] uses a classifier that is trained to distinguish between two datasets.

$\mathcal{A}$-distance is a popular measure in the literature, probably because it can be computed simply, has an intuitive meaning and is theoretically grounded.

Some recent works on domain adaptation have either used $\mathcal{A}$-distance in their analysis (e.g. Glorot et al. [50]) or have used it as a fundamental part of an adversarial learning approach (e.g. Ajakan et al. [4], Ganin et al. [47]).

2.2.3.2 Maximum Mean Discrepancy (MMD)

The concept behind Maximum Mean Discrepancy (MMD) is to take samples from the two domains in question and project the data in the samples into a Reproducing Kernel

[2]There is more discussion on L_p norms in Section 2.3.2.6

Hilbert Space (RKHS) using a kernel function. The MMD test statistic is the difference between the mean values of the domain samples computed in the RKHS. The smaller the test statistic the more likely it is that the two samples were drawn from the same domain i.e. the more similar the domains. Maximum Mean Discrepancy is defined in Borgwardt et al. [20] and Gretton et al. [57].

Under certain circumstances MMD is equivalent to the metric of Energy Distance [98]. Energy distance is a statistical distance between the distributions of random vectors, which characterizes equality of distributions [108].

MMD has been used in a variety of works on dataset shift and domain adaptation including Hoffman et al. [64], Long et al. [82], [83], Pan et al. [91]. Interestingly MMD has also been used for straightforward quantification but always under the *prior probability shift* assumption that class-conditional feature distributions are the same in both the Target and Source domains e.g. Iyer et al. [68], Kawakubo et al. [76].

2.2.3.3 Earth Mover Distance (EMD)

Earth Mover Distance (EMD) was first introduced by Rubner et al. [96] [62]. It is conceptually very similar to Wasserstein[3] distance. EMD and Wasserstein distance are the same when the two distributions being compared have equal mass [80]. EMD is popular measure for distributional similarity in the field of image processing (e.g. Rubner et al. [96]) but has also been used in other areas of dataset shift and domain adaptation (e.g. Hofer [62])

EMD is based on the concept of computing the minimal cost to transform one distribution to another and is effectively a transport problem [96]. As such the concept of ground distance is fundamental. In 2D images where pixels are features, ground distance has a natural physical meaning. In Hofer [62], Euclidean distance in the feature space is used as the measure for ground distance.

[3] Also known as Mallow distance

2.3 Unsupervised domain adaptation

Methods that address *dataset shift* typically go under the heading of *domain adaptation*. In this thesis we are only interested in *unsupervised dataset shift* i.e. when labelled data is *only* available from the Source domain and *not* from the Target domain.

Very commonly, researchers have been addressing situations where they have a classifier that has been trained with data from one domain and then want to perform the same classification task in a similar but non-identical domain where the amount of labelled data is limited or non-existent. There is the strong intuition that they should still try to use knowledge obtained from the original domain, but to adapt it to the new domain.

Domain adaptation is also referred to as *transductive transfer learning* itself a subset of *transfer learning* [90] [106]. Storkey [103] notes that 'the problem of dataset shift is closely related to another area of study known by various terms such as *transfer learning* or *inductive transfer*'.

Domain adaptation is of particular interest in the fields of natural language processing (NLP) and image processing.

Approaches to unsupervised domain adaptation can be broadly categorised into four groups:

- Mixtures of sub-domains
- Importance weighting of instances
- Feature representations
- Weakly supervised

Domain adaptation is a very large area of study. In this literature review I have focussed on selected papers that are particularly relevant to the quantification under class-conditional dataset shift problem.

2.3.1 Mixtures of sub-domains

With *mixture of sub-domain* methods, the assumption is that the Source and Target domains are both made up from a mixture of common sub-domains. While the class-conditional feature distribution differs between Source and Target domains the assumption

is that it does *not* vary within the sub-domains. The difference in class-conditional feature distribution between the Source and Target domains is then assumed to be fully accounted for by a difference in their constituent proportions of sub-domains.

2.3.1.1 Ensemble approaches

Broadly, in ensemble approaches, a classifier is trained specifically for each sub-domain and the overall output (say instance classification) is computed as a function of the outputs of the ensemble of classifiers. Works in this area include Mansour et al. [84] and Duan et al. [36].

2.3.1.2 Latent domains

In Alaiz-Rodríguez et al. [5] they extend the method from Saerens et al. [97] into 'sub-classes'. Within each class the class-conditional feature distributions $P(x|y)$ are assumed *not* to be the same in the Source and Target domains. Each class is considered to be made up of a number of sub-classes and for each sub-class the class-conditional feature distribution *is* assumed to be the same in both the Source and Target domains. The Saerens et al. [97] expectation-maximisation approach is applied at this sub-class level.

They reported some good results but when they ran the experiment using feature-based biassing, as used by Zadrozny [122] and Gretton et al. [57], they found that their method offered no improvement over the class-level method from Saerens et al. [97] and in some cases performed worse.

The method outlined in Hofer [62] is effectively a distribution matching method (see Section 2.1.3) but one which works at a latent sub-domain level. The conditional feature distributions from the Source domain and the unconditional feature distribution from the Target domain are separately modelled as mixtures of Gaussian distributions. The unconditional feature distribution in the Target domain is considered to be made up of probability mass transferred from the conditional feature distributions in the Source domain, where that transfer minimises the Earth Mover Distance (Section 2.2.3.3). Full details of the method are given in Section A.4.

The authors applied this method to a dataset of company insolvencies that was obtained from the Danish tax authority. Estimates of class-proportions were benchmarked against

estimates from two other baseline methods, *Global*: the Global Drift Model from the author's earlier work Hofer and Krempl [63] and *LFS*: the Linear Feature Shift model proposed in Biernacki et al. [17]. The published results indicate that the author's method is superior to the chosen baseline methods.

However, in deciding which of the many approaches to re-implement I rejected the Hofer [62] method for a number of reasons. The dataset on which it was tested was confidential so I could not get access to it. They had not applied their method to any public domain datasets. The dataset they used was very low dimensionality: 4 categorical and 2 continuous features in contrast to the higher dimensionality datasets in this thesis. None of the papers that have subsequently cited Hofer [62] have re-implemented the method. Finally the authors were approached but were unwilling to share their code.

Having said this, I still believe it would still be an interesting piece of further work to benchmark the results in this thesis against the method from Hofer [62].

2.3.2 Importance weighting of instances

The second general approach to unsupervised domain adaptation is *importance weighting of instances*, often shortened to *instance weighting*.

The principle behind instance weighting is to apply a weight to each instance of the training data that has been drawn from the Source domain so that its weighted joint probability distribution is as close as possible to that of the test data drawn from the Target domain. In theory, a classifier that is then trained on that weighted training data should perform well on the test data.

More detail on method for importance weighting of instances is given in Appendix B.

2.3.2.1 Sample selection bias correction

In Zadrozny [122], the training set is considered to be a sample drawn from the Target domain with a sampling bias based on the value of a selector variable. This sample selection bias is corrected for by applying weights to the instances in the training set. The weights are computed using the class-probabilities given by a classifier that has been trained to discriminate between instances from the training set and instances from the test set.

2.3.2.2 Kernel density estimation (KDE)

In common with other *instance weighting* approaches, importance weights to be applied to each instance in the training set are given by the ratio of the joint distributions:

$$w_i = \frac{P_T(x, y)}{P_S(x, y)} = \frac{P_T(y|x)P_T(x)}{P_S(y|x)P_S(x)}. \tag{2.7}$$

Again, in common with other *instance weighting* approaches, this method contains the assumption of *covariate shift* i.e. that $P_T(y|x) = P_S(y|x)$ so:

$$w_i = \frac{P_T(x_{Si})}{P_S(x_{Si})}. \tag{2.8}$$

The weights applied to the instances in the training set are the ratio of the data *density* in the two domains.

The most obvious approach is simply to independently estimate $P_S(x)$ and $P_T(x)$ using the training and test data. This was the approach taken by Shimodaira [101] in what was probably the first work to propose a method of instance weighting to deal with covariate shift. This method is efficient because it does not require optimisation.

However, Sugiyama et al. [105] describes this approach as 'naïve', as it suffers from the curse of high dimensionality and subsequently may not be reliable in high-dimensional problems. Sugiyama and Kawanabe [104] state that the method is contrary to Vapnik's principle [115] that: *one should not solve more difficult intermediate problems when solving a target problem.* They, and others, have proposed methods for direct estimation of the density ratio.

2.3.2.3 Kernel mean matching (KMM)

One such method for direct estimation of instance weights, Kernel Mean Matching, was first put forward by Huang et al. [66] and updated by the same authors in Gretton et al. [58]. KMM works by finding the weights for the training data that minimise the Maximum Mean Discrepancy between the weighted training data and the test data.

Huang et al. [66] reported that "KMM *always*[4] improves test performance compared to the unweighted case" but in Gretton et al. [58] the same authors report more ambiguous

[4] Author's italics

results. It appears that Huang et al. [66] had used overly simple models. Gretton et al. [58] found was that KMM did substantially improves learning performance in cases where the class of functions output by the learning algorithm is *simpler* than the true function. However, when better models were fitted to the data, KMM generally either did not affect performance or actually made it worse.

Further details of the Kernel Mean Matching method are given in Section B.2.

2.3.2.4 Unconstrained least-squares importance fitting (uLSIF)

Sugiyama and Kawanabe [104] put forward the Least Squares Importance Fitting (LSIF) method in which the instance weights w_i are given by:

$$w_i = \sum_{l=1}^{t} \alpha_l K_\sigma(\mathbf{x}_i, \mathbf{c}_l), \tag{2.9}$$

where K_σ is the Gaussian kernel function and $\mathbf{c}_l$ is a template point randomly chosen from the test set.

The vector of α values, $\boldsymbol{\alpha}$, is computed by minimising the squared loss between the densities in the Source and Target domains. LSIF is computationally very efficient but it sometimes suffers from a numerical problem and is therefore not reliable in practice. To address this problem, Sugiyama and Kawanabe [104] put forward the *unconstrained* version, uLSIF, where the non-negativity constraint on the α terms in the optimisation is replaced with a max function where $\alpha' = \max(0, \alpha)$.

Further details of the unconstrained Least-Squares Importance Fitting method are given in Section B.3

2.3.2.5 Comparison of importance weighting methods

Sugiyama and Kawanabe [104] evaluated several alternative methods for computing importance weights. These are set out in Table 2.5.

Table 2.5: Importance weighting methods from Sugiyama and Kawanabe [104]

Method	Title	Reference	Note
KDE	Kernel Density Estimation	Shimodaira [101]	Section 2.3.2.2
KMM	Kernel Mean Matching	Gretton et al. [57]	Section 2.3.2.3
LR	Logistic Regression		Also known as the log-linear model
KLIEP	Kullbeck-Leibler Importance Estimation Procedure	Sugiyama et al. [105]	
LSIF	Least Squares Importance Fitting	Kanamori et al. [74]	
uLSIF	Unconstrained Least Squares Importance Fitting	Kanamori et al. [74]	

Table 2.6 is a summary of their comparison of the different methods:

Table 2.6: Comparison of importance weighting methods [104]

Method	Density Estimation	Model Selection	Optimisation	Out-of-sample Prediction
KDE	Necessary	Available	Analytic	Possible
KMM	Not necessary	Not available	Convex QP	Not possible
LR	Not necessary	Available	Convex non-linear	Possible
KLIEP	Not necessary	Available	Convex non-linear	Possible
LSIF	Not necessary	Available	Convex QP	Possible
uLSIF	Not necessary	Available	Analytic	Possible

Sugiyama and Kawanabe [104] claim that uLSIF is a preferable method for importance estimation because it is solvable analytically (and is therefore fast) and enables parameter setting by cross-validation.

Bickel et al. [16] developed a method which computed instance weights *and* trained the classifier at the same time. They compared their method to the Gretton et al. [58] KMM method and to the Zadrozny [122] sample selection bias correction method. They found that their method improved performance on a spam filtering task whereas the Gretton et al. [58] KMM method degraded performance and the Zadrozny [122] sample selection

bias method only gave a marginal improvement. On a landmine detection task both their method and the Gretton et al. [58] KMM method performed well while the Zadrozny [122] sample selection bias correction method again gave only a very marginal improvement.

2.3.2.6 Distance measures in importance weighting

The KMM and uLSIF (and other) methods use a Gaussian kernel. The concept of *distance* is fundamental and the Gaussian kernels use Euclidean distance (the L_2 norm) in the standardised feature space. However Aggarwal et al. [3] notes that the meaningfulness of the L_k norm in high dimensional spaces is sensitive to the value of k. They found that the Manhattan distance metric (L_1 norm) is consistently preferable to the Euclidean distance metric (L_2 norm) for high dimensional data mining applications. High dimensionality is a concern in this thesis. The motivating example in which the feature set is accounts followed on Twitter is very high dimensional.

The exploration of alternative distance metrics is another interesting avenue for potential further work and is discussed in Chapter 7.

2.3.3 Feature representations

The third main approach to domain adaptation is to transform the features, x, so that information in the features that is relevant to classification is separated from, or unaffected by, information that relates to the domain from which the data was taken.

2.3.3.1 Feature manipulation

Structured Correspondence Learning (SCL) was put forward in Blitzer et al. [18]. They built on earlier work from Ando and Zhang [8]. Ben-David et al. [14] found that SCL was able to reduce the difference between the Source and Target domains. However they also observed that choosing the pivot features was potentially problematic [90]. In Blitzer et al. [19] SCL was updated to use mutual information for the selection of pivot features.

2.3.3.2 Subspace learning

The principle with *Subspace Learning* is to learn a new, typically lower-dimension, feature representation (a *subspace*) in which the difference between Source and Target domains is minimised but information required for the classification task is preserved.

Pan et al. [91] approach this as 'transfer learning via dimensionality reduction' . They chose Maximum Mean Discrepancy (MMD) (see Section 2.2.3.2) as their measure of distance between domains and named their method *Maximum Mean Discrepancy Embedding (MMDE)*. A kernel matrix K is computed by constrained optimisation to minimise the MMD between domains, then PCA is applied to the matrix K to construct the low-dimensional representation. Later, in Pan et al. [92] the authors note that this method has drawbacks, firstly that it is transductive and cannot generalise to unseen patterns, and secondly it is computationally intensive. They rework the method to use a much more efficient optimisation function and name this new method *Transfer Component Analysis*.

The methods in Gong et al. [51] and Gopalan et al. [55] involve creating a series of subspaces that follow a path that morphs the domain between between Source and the Target. This first part of the process is unsupervised. As a second step labelled data from the Source domain is then projected onto the sub-spaces and these projections used to train a classifier that can then be used to label data from the Target domain.

Fernando et al. [40] then builds on the work of Gong et al. [51] and Gopalan et al. [55]. Their approach is to first transform the Source and Target domains to respective subspaces by selecting their first d PCA components and then to find a mapping function that transforms the Source subspace into the Target subspace. The matrix that maps the Source to the Target subspace can be found in closed form. Their results on image processing tasks appear to compare favourably with others.

2.3.3.3 Unsupervised feature representation

As with Gong et al. [51] and Gopalan et al. [55] above, being *unsupervised* these methods do not *explicitly* attempt to generate a new domain-independent feature representation.

Raina et al. [94] utilised large quantities of unlabelled data to facilitate transfer learning in image processing in a method that they called *Self-Taught Learning*. Bengio et al. [15] used stacked autoencoders to generate a new feature representation. Vincent et al. [116] then

made a significant advance by switching from *stacked autoencoders* to *stacked de-noising autoencoders (SDA)*. Their intuition was that by inserting noise into the autoencoder it learns a mapping from input to output that is more robust. The motivation in Vincent et al. [116] was to generate a new feature representation which would improve the learning performance of deep neural network models. This method was then picked up specifically for domain adaptation in Glorot et al. [50] where it was tested experimentally on the Amazon product reviews dataset and benchmarked against various other state of the art domain adaptation methods including SCL [18]. They found that on their sentiment classification task the SDA method outperformed all other methods.

Chen et al. [29] put forward their *marginalised* version of the *Stacked De-noising Autoencoder*, the *mSDA* method, in which the SDA model parameters are computed quickly in closed form. Prior to this, in Glorot et al. [50] for example, the denoising autoencoders had been built with neural networks. Chen et al. [29] claim that while the mSDA method is faster by two orders of magnitude is achieves similar levels of performance to neural-network based SDAs.

In the mSDA architecture each autoencoder is simply a linear mapping $\mathrm{W} : \mathcal{R}^d \rightarrow \mathcal{R}^d$. Chen et al. [29] also believed that the non-linearity of the Glorot et al. [50] neural-network based SDA was key to their success. With the auto-encoding itself being linear, the non-linearity is added by including non-linear *squashing* functions between each layer of auto-encoder. Several options are available but Chen et al. [29] used the tanh() function. A further advantage of mSDAs is that they only require two parameters, the amount of noise to be added and the number of layers (typically up to 5). As the mSDA is fast to train, it is possible to set these parameters using cross validation.

While the mSDA method is better at dealing with high-dimension data than the SDA method, it is still a potential limitation. The authors include a method for performing mSDA on sub-sets of the features and recombining the results. This method is based on the works of Blitzer et al. [18] and Glorot et al. [50].

2.3.3.4 Adversarial feature representation

Adversarial feature representation methods have parallels with multi-task learning [22]. The common principle in these methods is that the new feature representation is learnt by simultaneously optimising two objectives: generating a representation that is *good* for

the required classification task but is also *bad* for the task of determining which domain the instance of data is from. Unlike the SDA method above, these methods learn domain adaption in *supervised* manner. These methods have a similarity to the *Subspace Learning* approaches outlined in Section 2.3.3.2.

Ganin and Lempitsky [46] [47] put forward the *Domain-Adversarial Neural Network (DANN)* method. As Chen et al. [28], Glorot et al. [50] and others had done before them they measured the performance of their approach on the Amazon reviews dataset. They used both the original features and a transformed feature set built using the *Marginalised Stacked Denoising Autoencoders* (mSDA) method[5] from Chen et al. [29].

According to their results, while DANN was better *most* of the time, some of the improvements look to be small. On the original features the mean accuracy across the Source-Target combinations went from 0.760 for the SVM baseline to 0.763. Looking at their results, pre-processing the features using the mSDA method appears to have a greater impact on performance than the DANN method itself.

Ganin and Lempitsky [46] [47] use proxy $\mathcal{A}$-distance as their measure of distance between domains while Shen et al. [100] suggest that the Wasserstein distance (see Section 2.2.3.3) is a better measure. Several other works ([83], [112] and [125] amongst others) use Maximum Mean Discrepancy (MMD) (see Section 2.2.3.2).

Published at a similar time to Ganin et al. [47], Zhuang et al. [127] shares their motivation of utilising a method that creates a feature representation that explicitly minimises the difference between domains, while simultaneously explicitly maximising information relating to labels. Like Vincent et al. [116], Glorot et al. [50] and Chen et al. [29] they base their method on auto-encoders. Where they differ is that in addition to the loss between the original input and its recreated output they also explicitly minimise loss functions in the two encoding hidden layers. On the first encoded layers they minimise the KL Divergence between an instance from each domain, minimising domain information in this encoded representation. On the second encoding layer they minimise the loss on a softmax label classifier.

Tzeng et al. [113] follow a similar adversarial approach, using a deep model and optimising a loss function that includes both *domain confusion* loss (which seeks to make the domains

[5] 5 layers, 50% noise

indistinguishable) *and* classification loss on the labelled data. Their approach is to use a small amount of labelled data in the Target domain (i.e. it is supervised) to ensure alignment over classes. They make an interesting observation that while maximising domain confusion pulls the marginal distributions of the domains together, it does not necessarily align the classes in the Target with those in the Source. This touches on the point made earlier that for good quantification we expect a difference in class distribution between Source and Target, but aim for as little difference as possible in class-conditional feature distribution.

In Tzeng et al. [114] the authors have tried to put the adversarial methods for domain adaptation into a generalised framework. By generalising the methods of others they claim to have arrived at a novel configuration which they call *Adversarial Discriminative Domain Adaptation (ADDA)*.

2.3.4 Weakly supervised

Weakly supervised approaches to domain adaptation are not as prominent in the field of domain adaptation and are included here for completeness. In weakly supervised approaches, instances in the test set are given class probabilities by a classifier trained on the training set. The instances with the highest probability of being in a particular class are assigned that class label and are added to the training set with some weighting. Zhou [126] gives a broad overview of the field of *weakly supervised learning* but in this thesis I have just focussed on the approach set out by Jiang and Zhai [71]. Their approach was iterative with a number of instances being transferred from the test set to the training set on each iteration before re-training the classifier with the revised training set. They found that giving a higher weight to the instances transferred from the test domain gave a higher performance.

2.4 Quantification under class-conditional dataset shift

As stated in the Introduction, what makes this thesis novel is that there has been very little published work on quantification under conditions of class-conditional dataset shift. In their 2017 review paper *A review of quantification learning* González et al. [52] states that 'Only a few methods assume that $P(x|y)$ may change, for instance Hofer [62]'. Hofer

[62]' is reviewed in Section 2.3.1.2 and in more detail in A.4.

Chapter 3

Domain adaptation with explicit sub-domains

As discussed in Section 2.3.1, a number of authors have approached the problem by considering the Source and Target domains to be a mixture of common sub-domains in which the class-conditional feature distributions are domain-invariant. Typically these authors assume that the sub-domains are latent. In this Chapter, a simpler approach is taken where it is assumed that the sub-domains are *explicitly* labelled in the training data.

If the class-conditional feature distributions at a sub-domain level are domain-invariant then a standard *classify and adjust* quantifier for each sub-domain should be effective for estimating the class proportions in that sub-domain. The outputs from each of these sub-domain quantifiers can then be aggregated to give an estimate of overall class proportions in the Target domain.

Going back to Chapter 1 and the motivating example of Twitter users tweeting about retirement homes in Scotland. The *sub-domains* could possibly be by age: older vs. younger, by location: Scotland vs. the rest of the UK or maybe by both: older Scottish people vs. the rest of the UK. *Class-conditional dataset shift* in this context is assumed to be fully accounted for by a difference in the proportion of the groups in our test set to the proportion which we originally had in our training set. Given the features that we are using in our motivating example, Twitter accounts followed, this makes intuitive sense. Older people are unlikely to follow Ariana Grande or the latest YouTube vloggers. Southern Rail's Twitter account is unlikely to be of interest to many Scots.

Switching to the more formal language of data science, the question addressed in this chapter is: *when class-conditional dataset shift has occurred, can we get better quantification accuracy by considering the Source and Target domains to be made up from a mixture of common sub-domains where we can assume that conditional dataset shift has not occurred?*

The work in this chapter rests on four assumptions:

1. The class-conditional feature distributions within each sub-domain do not vary with domain i.e. $P_S(x|y, sd) = P_T(x|y, sd)$, where sd designates sub-domain.
2. Each instance is from one sub-domain only.
3. The sub-domain for each instance in the Source domain is known and labelled and hence is *explicit*.
4. The difference in class-conditional feature distribution at Source and Target domain level is completely accounted for by different proportions of the sub-domains in those domains.

The approach in this chapter is limited by the assumption that the sub-domain is known. It may be a relatively simple task to identify the sub-domain that is the main source of the difference in class-conditional feature distribution between the Source and Target domains. In the example given above, age and location are clearly reasonable candidates. However, in other cases, determining the right sub-domain may not be straightforward.

The first of the four assumptions is that $P_S(x|y, sd) = P_T(x|y, sd)$. For a discriminative classifier $P(\hat{y}|x)$ is constant i.e. once the classifier is trained the probability of assigning an estimated label $\hat{y}$ is *solely* dependent on the features x that are input to the classifier. If $P(x|y, sd)$ is constant then so therefore is $P(\hat{y}|y, sd)$.

$P(\hat{y} = 0|y = 0)$ is the *recall* for class 0 (r_0) while $P(\hat{y} = 1|y = 1)$ is the *recall* for class 1 (r_1). If we designate class 0 as the *positive* class and class 1 as the *negative* class then the relationship between *recall*, *tpr* and *fpr* is simply:

$$r_0 = tpr, \tag{3.1}$$

$$r_1 = 1 - fpr. \tag{3.2}$$

If $P(\hat{y}|y, sd)$ is constant then *recall* is constant for each combination of main-class and sub-domain. This is a core assumption in this chapter.

A first step was to run some exploratory experiments using real Twitter data. These are described in Section 3.4. These showed that while using sub-domains can reduce the *bias* in the estimate of class proportions it can also increase the level of noise, leading to an increase the *variance*. The problem of bias vs. variance in quantification with explicit sub-domains looked like a problem that might be solvable in closed-form. This was explored in Sections 3.5 and 3.6.

A closed-form solution was found to only be realistic when we assume we have a large set of validation data from the Source domain so in Section 3.8 numerical simulation was used to get past this limitation and explore the problem more generally.

3.1 Definitions

It is important to first establish some definitions that will be used throughout the chapter.

If we have a dataset with two classes, 0 and 1. The number of instances in class 0 and 1 (*actual* counts) are given by a_0 and a_1 respectively and we define $\mathbf{a}$ as the vector of those counts:

$$\mathbf{a} = \begin{pmatrix} a_0 \\ a_1 \end{pmatrix}. \tag{3.3}$$

The classifier assigns a predicted class label to each instance. The number of instances by predicted class for class 0 and class 1 are given by p_0 and p_1 respectively and we define $\mathbf{p}$ as the vector of those counts:

$$\mathbf{p} = \begin{pmatrix} p_0 \\ p_1 \end{pmatrix}. \tag{3.4}$$

The classifier recall for classes 0 and 1 are given by r_0 and r_1. By definition these recall values relate the values (a_0, a_1) to (p_0, p_1):

$$\begin{pmatrix} p_0 \\ p_1 \end{pmatrix} = \begin{pmatrix} r_0 & (1-r_1) \\ (1-r_0) & r_1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix}. \tag{3.5}$$

We define this matrix of recall values as R:

$$\mathrm{R} = \begin{pmatrix} r_0 & (1-r_1) \\ (1-r_0) & r_1 \end{pmatrix}, \tag{3.6}$$

so:

$$\mathbf{p} = \mathrm{R}\mathbf{a}. \tag{3.7}$$

Going further, if we define the matrix P as our confusion matrix:

$$\mathrm{P} = \begin{pmatrix} p_{00} & p_{01} \\ p_{10} & p_{11} \end{pmatrix}, \tag{3.8}$$

where p_{ij} is the count of the number of instances of actual class j that have been assigned class label i by the classifier.

From the definition of **p**:

$$\mathbf{p} = \mathrm{P}\mathbf{1}_n, \tag{3.9}$$

and from the definition of **a**:

$$\mathbf{a} = \mathrm{P}^T\mathbf{1}_n, \tag{3.10}$$

where $\mathbf{1}_n$ is a vector of 1's of size n, and n is the number of classes. In this case n=2:

$$\mathbf{1}_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}. \tag{3.11}$$

If we convert **a** into a diagonal matrix A:

$$\mathrm{A} = \begin{pmatrix} a_0 & 0 \\ 0 & a_1 \end{pmatrix}, \tag{3.12}$$

then:

$$\mathrm{P} = \mathrm{RA}, \tag{3.13}$$

i.e.:

$$\begin{pmatrix} p_{00} & p_{01} \\ p_{10} & p_{11} \end{pmatrix} = \begin{pmatrix} r_0 & (1-r_1) \\ (1-r_0) & r_1 \end{pmatrix} \begin{pmatrix} a_0 & 0 \\ 0 & a_1 \end{pmatrix}. \tag{3.14}$$

So we can derive the R matrix from the confusion matrix P and the counts by actual class A:

$$\mathrm{R} = \mathrm{P}(\mathrm{A})^{-1}. \tag{3.15}$$

3.1.1 Validation data

We assume we have a validation set of data from the Source domain that is labelled *both* for main-class *and* for sub-domain and has *not* been used in the training of the classifier. In practice this would typically be achieved with cross-validation on the training set. The validation set is used to determine the performance of the classifier, in this case to compute the recall values.

The subscript v is used to denote *validation*. The count by actual class $\mathbf{a}_v$ (and A_v) is given for this dataset. This dataset is classified by the trained classifier which assigns a predicted class label to each instance. The R matrix computed from the validation data, R_v, is then derived from the resulting confusion matrix P_v and from A_v:

$$\mathrm{R}_v = \mathrm{P}_v(\mathrm{A}_v)^{-1}. \tag{3.16}$$

3.1.2 Test data

The aim of quantification is to estimate the counts by actual class in the test set designated as $\hat{\mathbf{a}}_t$.

We define the error $\mathbf{e}_t$ as the difference between our estimated class distribution in the test set $\hat{\mathbf{a}}_t$ and the actual class distribution in the test set $\mathbf{a}_t$ i.e.:

$$\mathbf{e}_t = \hat{\mathbf{a}}_t - \mathbf{a}_t. \tag{3.17}$$

3.1.3 Extension to sub-domains

In the introduction to this chapter we made the assumption that recall is constant for each combination of class and sub-domain. To avoid confusion over terminology the original 2-classes are now renamed as the *main* classes and are designated as α and β. The two sub-domains are designated as γ and δ.

Our 2 main-class, 2 sub-domain problem is now converted to a 4-class problem where each combination of main-class and sub-domain is a separate class:

Table 3.1: Class, main-class and sub-domain

Class	Main-class	Sub-domain
1	α	γ
2	α	δ
3	β	γ
4	β	δ

In this example there are 2 main-classes and 2 sub-domains but clearly this approach can be extended to problems with any number of classes and sub-domains.

3.1.4 R*

R^* is defined as the matrix of *latent* recall probabilities for a classifier on a domain. The elements in the R^* matrix, r_{ij}, are the probabilities with which a randomly sampled instance from class j is assigned the estimated class label of i.

For a set of instances, sampled iid from the domain and classified by a trained classifier, the resulting number of instances by predicted class (given by the vector $\mathbf{p}$) is stochastic with its values determined by the actual number of instances in each class given in $\mathbf{a}$ and the latent recall probabilities in R^* for the classifier on that domain.

As a stochastic process, if we take multiple iid samples from the domain each with the same counts by actual class given in $\mathbf{a}$ then we will obtain a distribution of values of $\mathbf{p}$.

$$\mathbf{p} \sim R^*\mathbf{a}. \tag{3.18}$$

This distribution will be multinomial (binomial in the case where there are only two classes).

Each separate set of data k of actual class count $\mathbf{a}$ will, when processed by the classifier, generate a vector of predicted class counts $\mathbf{p}_k$ which we can connect with a matrix R_k. We can consider R_k to be the matrix of *observed* recall values for data sample k:

$$\mathbf{p}_k = R_k\mathbf{a}. \tag{3.19}$$

If the instances are sampled iid from the domain then, according to the Law of Large Numbers, as the size of the sample increases then the *observed* R_k will tend towards the *latent* R^*.

3.1.5 Quantification performance measures

Two measures of quantification error are used in this thesis: Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE).

If e_i are the model errors:

$$e_i = \hat{m} - m, \tag{3.20}$$

where m is the actual class proportion and $\hat{m}$ is the estimated class proportion, then Absolute Error (AE) is defined as:

$$\text{AE} = |e_i|, \tag{3.21}$$

and for a set of n model errors, Mean Absolute Error is simply:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|e_i|, \tag{3.22}$$

and Root Mean Squared Error (RMSE) [23] is:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}e_i^2}. \tag{3.23}$$

RMSE can be decomposed into *bias* and *variance*:

$$\text{RMSE} = \sqrt{bias^2 + variance}. \tag{3.24}$$

3.2 Quantification by matrix-inversion

Equation 3.7 states:

$$\mathbf{p} = \mathrm{R}\mathbf{a}, \tag{3.25}$$

so:

$$\mathbf{a} = \mathrm{R}^{-1}\mathbf{p}. \tag{3.26}$$

This applies to both the test set:

$$\mathbf{a}_t = \mathrm{R}_t^{-1}\mathbf{p}_t, \tag{3.27}$$

and to the validation set:

$$\mathbf{a}_v = \mathrm{R}_v^{-1}\mathbf{p}_v. \tag{3.28}$$

Clearly, R_t is *unknown* and we want to estimate $\mathbf{a}_t$. To compute our estimate $\hat{\mathbf{a}}_t$ we *assume* that we can use our *known* R_v^{-1} in place of the *unknown* R_t^{-1}.

If we make that assumption then we define our estimate of $\mathbf{a}_t$ as $\hat{\mathbf{a}}_t$ where:

$$\hat{\mathbf{a}}_t = \mathrm{R}_v^{-1}\mathbf{p}_t. \tag{3.29}$$

Solving such an *inverse problem* by inverting a matrix can sometimes be problematic. However, matrix-inversion makes sense in our case because:

- R is square
- With a limited number of classes the R matrix is small

- The R matrix will normally be conditioned: when classifying instances of class i a realistic classifier will typically predict class i more than other classes. So the R matrix will normally have large values on its leading diagonal and smaller values elsewhere.

Importantly, matrix-inversion allows us to explore a closed-form solution.

The matrix-inversion method is the method used by Vucetic and Obradovic [117] and Forman [44] amongst others. The equivalence of matrix-inversion and Forman's Adjusted Count method [44] is shown in Section A.1

3.3 Quantification with and without sub-domains

The method that uses sub-domains is defined as the *sd-method* and the method that does *not* use sub-domains as the *nsd-method*. We want to compare these two methods on the *same* data.

3.3.1 Method

The common data is created with both main-classes and sub-domains. The sd-method works with sub-domains throughout the process, only marginalising them out at the end to get the estimates by main-class. The nsd-method ignores the sub-domains in the data by marginalising them out at the start.

The process for comparing the sd and nsd methods on the same data is shown diagrammatically in Figure 3.1:

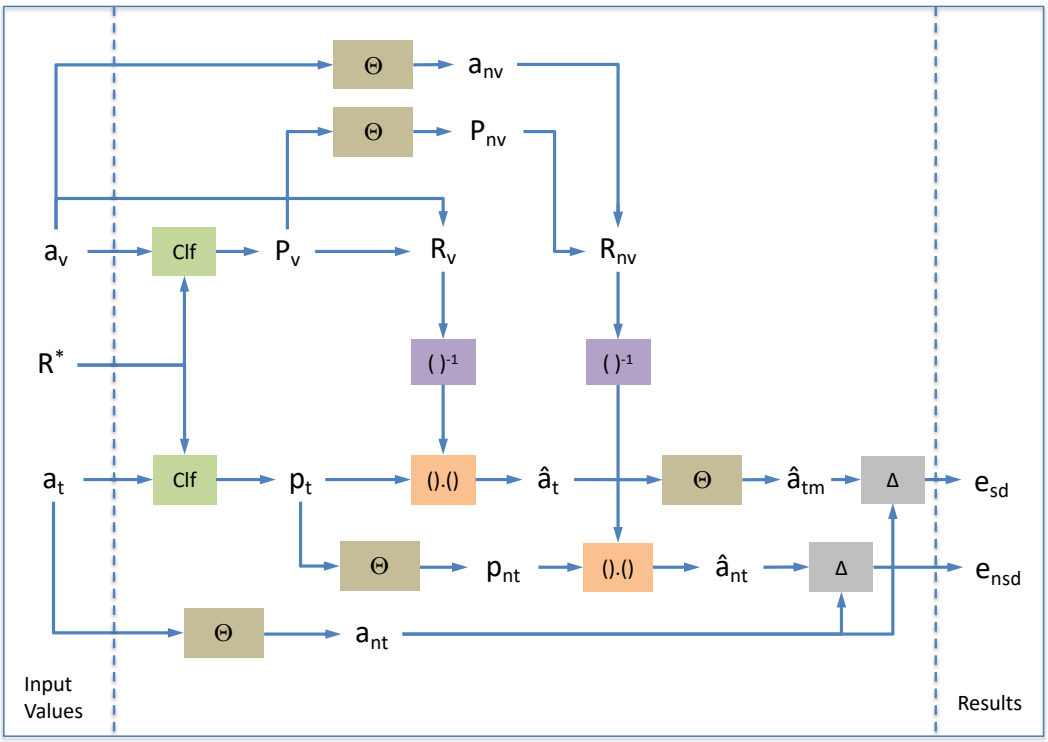

Figure 3.1: Process for computation of quantification error with and without the use of sub-domains. See Table 3.2 for key to symbols.

Table 3.2: Key to symbols

Clf	Classify the dataset	().()	Matrix-vector dot-product multiplication
$()^{-1}$	Matrix-inversion	Θ	Marginalise-out the sub-domains
Δ	Difference		

Considering again our 2-class, 2-sub-domain problem as a 4-class problem:

Table 3.3: Class, main-class and sub-domain

Class	Main-class	Sub-domain
1	α	γ
2	α	δ
3	β	γ
4	β	δ

The trained classifier classifies the validation set (labelled for both main-class and sub-

domain) generating a confusion matrix P_v:

$$\mathrm{P}_v = \begin{pmatrix} p_{v11} & p_{v12} & p_{v13} & p_{v14} \\ p_{v21} & p_{v22} & p_{v23} & p_{v24} \\ p_{v31} & p_{v32} & p_{v33} & p_{v34} \\ p_{v41} & p_{v42} & p_{v43} & p_{v44} \end{pmatrix}, \tag{3.30}$$

where p_{vij} is the count of the number of instances in the validation set of actual class j that have been assigned class label i by the classifier.

3.3.1.1 With sub-domains (sd method)

$\mathbf{a}_v$ is the vector of counts by class in the validation set. The matrix A_v is $\mathbf{a}_v$ in diagonalised form. R_v is given by Equation 3.16:

$$\mathrm{R}_v = \mathrm{P}_v(\mathrm{A}_v)^{-1}. \tag{3.31}$$

$\mathbf{a}_t$ is the vector of counts by class in the test set. It is classified by the same trained classifier to give a vector of counts by predicted class of $\mathbf{p}_t$. The estimate of the counts by actual class $\hat{\mathbf{a}}_t$ is given by Equation 3.29:

$$\hat{\mathbf{a}}_t = \mathrm{R}_v^{-1}\mathbf{p}_t. \tag{3.32}$$

Finally, at the *end* of the process, the sub-domains are marginalised out to get our estimate of counts by actual *main-class* $\hat{\boldsymbol{a}}_{tm}$:

$$\hat{\boldsymbol{a}}_{tm} = \begin{pmatrix} \hat{a}_{t\alpha} \\ \hat{a}_{t\beta} \end{pmatrix} = \Theta^T(\mathrm{R}_v)^{-1}\boldsymbol{p}_t = \Theta^T\mathrm{A}_v(\mathrm{P}_v)^{-1}\boldsymbol{p}_t, \tag{3.33}$$

where $\hat{a}_{t\alpha}$ and $\hat{a}_{t\beta}$ are the estimated number of instances in the test set in main-class α and β respectively and where the *marginalising matrix* Θ is given by:

$$\Theta = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}. \tag{3.34}$$

3.3.1.2 Without sub-domains (nsd method)

Both methods are applied to the same underlying data so A_v, P_v and $\mathbf{p}_t$ are all the same as above, but the sub-domains are marginalised out at the *start* of the process.

We define the 2x2 matrix P_{nv} as:

$$\mathrm{P}_{nv} = \begin{pmatrix} p_{v\alpha\alpha} & p_{v\alpha\beta} \\ p_{v\beta\alpha} & p_{v\beta\beta} \end{pmatrix}, \tag{3.35}$$

where p_{vij} is the count of the number of instances in the validation set of main-class j that have been assigned a main-class label i by the classifier. This is produced by marginalising out the sub-domains from P_v with the marginalising matrix Θ:

$$\mathrm{P}_{nv} = \Theta^T \mathrm{P}_v \Theta. \tag{3.36}$$

Similarly the sub-domains are marginalised from A_v to give A_{nv}:

$$\mathrm{A}_{nv} = \begin{pmatrix} a_{v\alpha} & 0 \\ 0 & a_{v\beta} \end{pmatrix} = \Theta^T \mathrm{A}_v \Theta. \tag{3.37}$$

From Equation 3.15 we can define the 2x2 matrix R_{nv} in terms of the 2x2 matrices P_{nv} and A_{nv}:

$$\mathrm{R}_{nv} = \mathrm{P}_{nv}(\mathrm{A}_{nv})^{-1}. \tag{3.38}$$

Substituting in the expressions for P_{nv} and A_{nv} from Equations 3.35 and 3.37 then gives:

$$\mathrm{R}_{nv} = \Theta^T \mathrm{P}_v \Theta (\Theta^T \mathrm{A}_v \Theta)^{-1}. \tag{3.39}$$

Similarly we are disregarding sub-domain information in our counts by predicted class $\mathbf{p}_t$ to give a vector of counts by predicted main-class only $\mathbf{p}_{nt}$:

$$\mathbf{p}_{nt} = \begin{pmatrix} p_{t\alpha} \\ p_{t\beta} \end{pmatrix} = \Theta^T \mathbf{p}_t, \tag{3.40}$$

and the estimate of class distribution by main-class in our test set is now computed, again using Equation 3.16:

$$\hat{\mathbf{a}}_{nt} = \mathrm{R}_{nv}^{-1} \mathbf{p}_{nt}, \tag{3.41}$$

so when we do *not* use sub-domains our estimate of actual counts by main-class is:

$$\hat{\mathbf{a}}_{nt} = \Theta^T \mathrm{A}_v \Theta (\Theta^T \mathrm{P}_v \Theta)^{-1} \Theta^T \mathbf{p}_t. \tag{3.42}$$

Comparing this to the expression for the class estimate when *using* sub-domains given in Equation 3.43:

$$\hat{\boldsymbol{a}}_{tm} = \Theta^T \mathrm{A}_v \mathrm{P}_v^{-1} \boldsymbol{p}_t, \tag{3.43}$$

then the difference in the estimates between the sd-method and the nsd-method is given by:

$$\hat{\boldsymbol{a}}_{tm} - \hat{\boldsymbol{a}}_{nt} = \Theta^T \mathrm{A}_v \big(\mathrm{P}_v^{-1} - \Theta (\Theta^T \mathrm{P}_v \Theta)^{-1} \Theta^T \big) \mathbf{p}_t. \tag{3.44}$$

3.4 Initial experiment

The aim of this initial experiment was to explore the sd and nsd methods with a real dataset.

3.4.1 Dataset

This initial experiment used the 4dSDA dataset of 5,981 Twitter users (instances) that was split into 2,925 for training and 3,016 for testing. The dataset was labelled for age group (main-class) and gender (sub-domain).

The 4dSDA dataset is an aggregation of the 4dUser dataset and the SDA dataset.

3.4.1.1 4dUser

Twitter accounts were sampled using the free random 1% tweet feed using Method52[1]. Users were then filtered by Language ('en') and Timezone (London or Edinburgh). The hypothesis was that in order to make a chosen screenname[2] unique, some users would add the four digits of their date of birth at the end. Assuming that this was the case, we filtered for only users with screennames where the last four digits were numerical in the range 1949 to 1999. Further screening was made to remove users that did not follow any other Twitter accounts ('friends') or had a very large number of followers (which were likely to be organisations or celebrities rather than 'normal' individuals).

3.4.1.2 SDA

Chris Inskip at The University of Sussex [67] generated the SDA dataset of Twitter users. He sampled the Twitter 1% tweet feed and applied similar filtering to that used in the creation of the 4dUser dataset. He then used a set of regular expressions ('reg-exes') to extract declarations of age from user's description text (e.g. '...I am a 21 year old student at...'). Some users may be less conscientious than others about maintaining their Twitter user description so it was assumed that some age descriptions may be a little out of date.

[1] Courtesy of CASM Consulting LLP

[2] Each user creates their own 'screenname' which must be unique

3.4.1.3 Aggregation and validation

Through the Polly project we had access to a dataset of 2,749 individuals containing both Twitter screennames and a range of demographic information including age (the 'MR[3] dataset'). The dataset was small and we had concerns that it may not be very representative, but it gave us ground-truth age labels for a set of Twitter users that we could use for validating the other datasets.

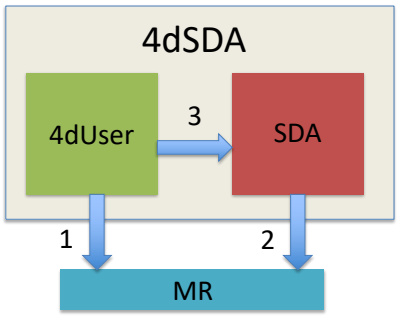

Figure 3.2: Validation of the 4dSDA dataset between datasets

Validation 1: 79 users in the MR dataset had Twitter screennames that met the criteria for the 4dUser dataset. In 77 of these cases the implied year of birth from the screenname corresponded to the given age of the individual.

Validation 2: 16 users in the MR dataset generated an age using the process used to create the SDA dataset. In 12 cases this was exactly correct and in a further 3 it was correct within 3 years.

Validation 3: 86 users in the SDA dataset met the criteria for the 4dUser dataset. 76 of these gave an exact age match. A further 3 were within 4 years. Of the remaining 7 cases, after inspecting the Twitter accounts it appears that 4dUser was correct with 2 and SDA with the other 5.

I felt that the validation was strong enough to combine the 4dUser and SDA datasets into a single 4dSDA dataset for use.

[3]Market Research

3.4.1.4 Age labels

Instances with an estimated year of birth of 1983 or before were labelled 0 and instances with an estimated year of birth from 1991 onwards were labelled 1. Instances with an estimated year of birth 1984-1990 inclusive were discarded. The distribution of the Twitter users in the 4dSDA dataset by estimated year of birth is shown in Figure 3.3.

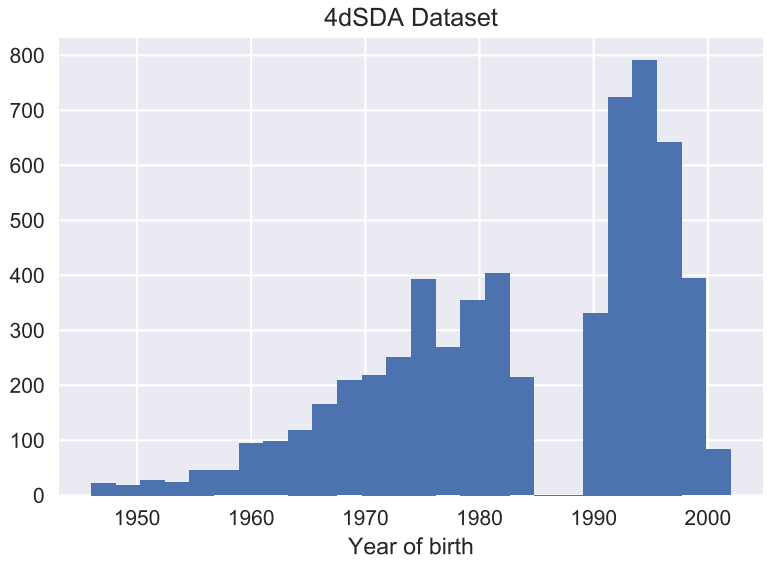

Figure 3.3: Distribution by estimated year of birth in the 4dSDA dataset

3.4.1.5 Gender labels

Thomas Kober of the University of Sussex applied gender labels to the datasets. He mainly used online tools where gender is assigned based on the first word in the name field, typically based on a person's first name. Instances where this word would not resolve to a gender id were removed from the dataset e.g. organisation names, blank fields or non gender-specific names such as 'Alex'. Of the 5,981 instances in the final dataset 4,139 were labelled male and 1,802 labelled female.

3.4.2 Method

The binary age labels were used to label main-class and the binary gender labels were used to label sub-domain. Two classifiers were trained on the labelled training data, one for main-class and one for sub-domain. The features were the IDs of the Twitter accounts

that each user followed.

3.4.2.1 Algorithm

The pseudocode for the simulation program is given below.

Data: 4dSDA
Result: Actual and estimated test set class proportions
while *split count* < 40 **do**
  split the dataset into *training* and *validation-test*;
  for *main-class proportion in* $[0.0, 0.2, 0.4, 0.6, 0.8, 1.0]$ **do**
    for *sub-domain proportion in* $[0.0, 0.2, 0.4, 0.6, 0.8, 1.0]$ **do**
      while *training count* < 10 **do**
        sample a class-balanced training set of 1600 instances from *training* split;
        train main-class and sub-domain classifiers;
        classify all instances in *validation-test* split;
        while *validation count* < 10 **do**
          sample a class-balanced validation set of 1000 instances from *validation-test* split;
          compute R matrices for sd and nsd;
        end
        while *test count* < 10 **do**
          sample a test set of 800 instances of given main-class and sub-domain proportions from *validation-test* split;
          compute counts by predicted main-class for nsd;
          compute counts by predicted main-class and sub-domain for sd;
        end
        compute estimated class proportions in test with sd and nsd method for every validation count / test count combination;
      end
    end
  end
end

Algorithm 1: Initial explicit sub-domains experiment

3.4.3 Results

3.4.3.1 Classify and count

Figure 3.4 shows the result of calculating class proportions by simply counting up the instances by predicted class:

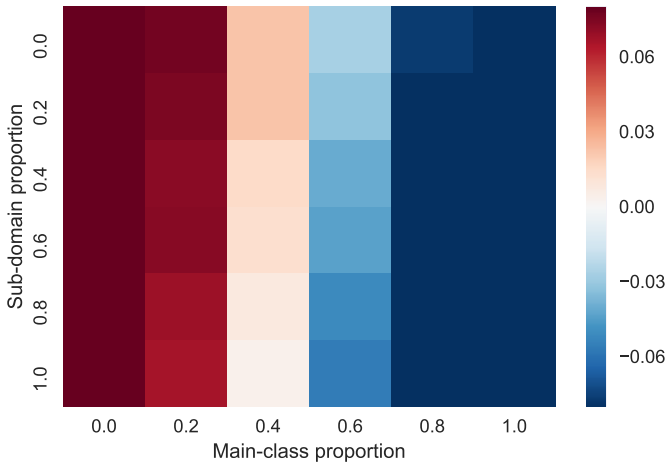

Figure 3.4: Mean main-class proportion error vs. main-class proportion and sub-domain proportion. *Classify and count* method. 4dSDA dataset

The result is as expected. The imperfect nature of the classifier leads to over and under-estimation of class proportions at the extremes with some cross-over point where true and predicted class proportion is the same.

3.4.3.2 Classify and adjust without sub-domains (nsd-method)

Figure 3.5 shows the result of applying the *classify and adjust* method *without* using sub-domains:

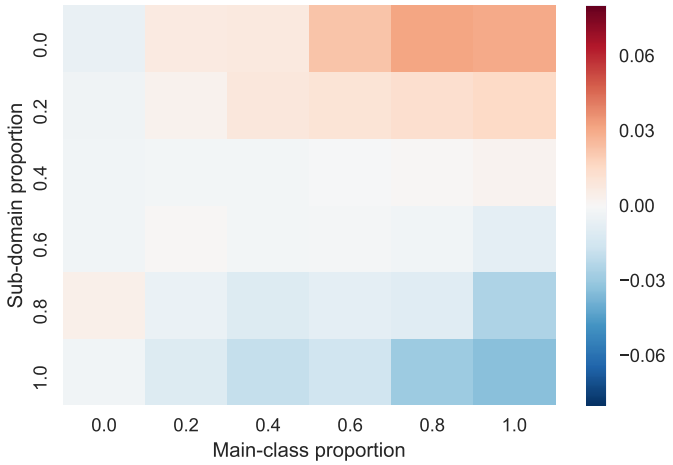

Figure 3.5: Mean main-class proportion error vs. main-class proportion and sub-domain proportion. nsd-method. 4dSDA dataset

Applying the standard *Adjusted Count* formula using the computed recall values by main-class without using sub-domains gives a more accurate mean estimate of class proportions than the *classify and count* approach shown in Figure 3.4.

Figure 3.5 also shows that when the dataset is predominantly main-class 0 (the right hand side of the chart) then the level of class proportion estimation error is sensitive to the sub-domain proportions of the sample. This is because recall for main-class 0 varies between sub-domains much more than recall varies for mainclass 1.

Table 3.4: Observed recall values by main-class and sub-domain in the 4dSDA dataset

	sub-domain 0	sub-domain 1	Difference
Recall main-class 0	0.863	0.880	0.017
Recall main-class 1	0.836	0.841	0.005

3.4.3.3 Classify and adjust with sub-domains (sd-method)

Figure 3.6 shows the mean main-class proportion error against main-class proportion and sub-domain proportion using the sd-method:

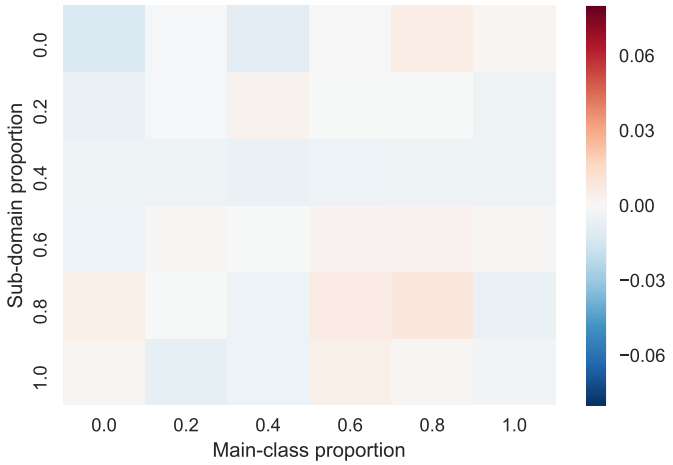

Figure 3.6: Mean main-class proportion error vs. main-class proportion and sub-domain proportion. sd-method. 4dSDA dataset

By using sub-domains, the estimation bias in mean values observed with the nsd-method largely disappears.

3.4.3.4 Comparison of methods

The heatmaps shown above give *mean* errors computed from repeated samples. Each of the main-class-sub-domain combination has been estimated 40,000 times. Variance has been averaged out.

However, when looking at Root Mean Squared Error (RMSE) the sd-method actually gives a slightly *higher* error than the nsd-method.

Table 3.5: RMSE of sd and nsd methods

Method	RMSE	2.5% CI	97.5% CI
sd	0.041	0.04054	0.04065
nsd	0.038	0.03765	0.03775

RMSE can be de-composed into *bias* and *variance* (Equation 3.24). Figures 3.7 and 3.8 are from the the same data as Figures 3.5 and 3.6.

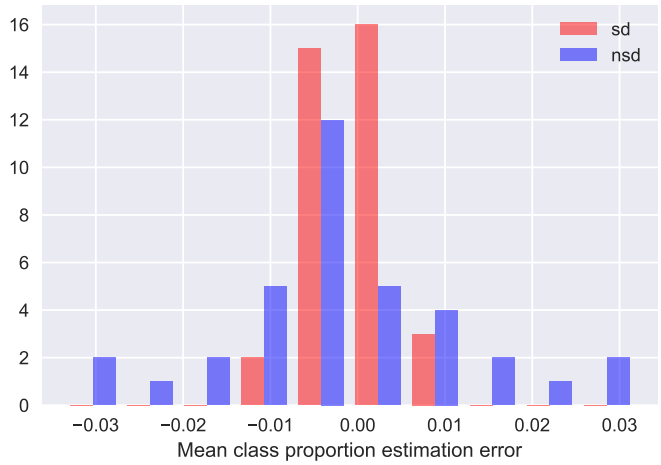

Figure 3.7: Mean value of class proportion estimate error

Figures 3.7 shows that *bias* is lower with the sd-method than with the nsd-method. This is as would be expected from Figures 3.5 and 3.6.

Figure 3.8 shows the distribution of *variance* values for the results using the sd-method and the nsd-method.

Figure 3.8: Variance in class proportion estimate error

Variance is typically *higher* with the sd-method than with the nsd-method.

Table 3.6 gives the decomposition of quantification error:

Table 3.6: Mean and variance sd and nsd method

Method	Variance of Mean x10e-6	Mean of Variance x10e-3
sd	19	1.63
nsd	193	1.23

On average, the sd-method has lower *bias* but higher *variance* than the nsd-method.

3.4.4 Discussion

This experiment on the 4dSDA dataset showed that using explicit sub-domains reduced the bias (mean error) in the estimate of the class proportions in the test set. However, while the bias was reduced the variance increased. Overall the Root Mean Squared Error (RMSE) was slightly *higher* when using explicit sub-domains (sd method) than when *not* using explicit sub-domains (nsd method).

In other circumstances the reduction in *bias* could outweigh any increase in *variance* meaning that the sd method would give a *lower* overall error (e.g. as measured by RMSE) than the nsd-method.

The next section explores whether it is possible to compute expected RMSE from the sd and nsd methods analytically in closed-form.

3.5 Analytic exploration

The initial experiment in Section 3.4 showed that using explicit sub-domains can reduce *bias* but increase *variance*. Given that overall quantification error expressed as RMSE is a combination of both bias and variance then it is important to understand whether using explicit sub-domains will *reduce* or *increase* that overall error.

The aim of this section is to see if that assessment can be made analytically. To see if a closed-form expression can be derived for quantification error.

As discussed in Section 3.1.4, under the assumptions made in this chapter the distribution

of the counts by predicted class can be considered as a multinomial distribution. Multinomial distributions can be approximated to the normal (Gaussian) distribution [99]. The sum of normal random variables is normal and the product of normal distributions is normal. Given this, it is reasonable to assume that it *may be possible* to derive closed-form expressions for quantification error, although none has been found in the papers on quantification. Vucetic and Obradovic [117] observed that while the distributions of predicted values could easily be estimated using the multinomial distribution it can be difficult to obtain the distribution of the estimated true prevalence in a closed form.

3.5.1 Classes

As set out previously in Table 3.1, the 4 classes are defined by the 2 main-classes and 2 sub-domains:

Table 3.7: Class, main-class and sub-domain

Class	Main-class	Sub-domain
1	α	γ
2	α	δ
3	β	γ
4	β	δ

3.5.2 Random variables

Equation 3.29 gives us the estimated counts by actual class $\hat{\mathbf{a}}_t$ as:

$$\hat{\mathbf{a}}_t = \mathrm{R}_v^{-1}\mathbf{p}_t, \tag{3.45}$$

and Equation 3.16 gives us R_v as:

$$\mathrm{R}_v = \mathrm{P}_v(\mathrm{A}_v)^{-1}. \tag{3.46}$$

We can consider the confusion matrix P to be a matrix of random variables P_{ij} where, for a dataset with actual counts by class $\mathbf{a}$, each random variable P_{ij} captures a probability distribution for the number of instances of actual class j that are assigned the predicted

class of i:

$$\mathrm{P} = \begin{pmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \\ P_{41} & P_{42} & P_{43} & P_{44} \end{pmatrix}. \tag{3.47}$$

A_v is a matrix with scalar values on its leading diagonal and zeros otherwise so, like P_v, R_v will be a matrix of random variables.

For each dataset the counts by actual class, $\mathbf{a}$ and A are fixed values. These values are *known* for the validation set and *unknown* for the test set (our aim is of course to estimate these for the test set). As set out in Section 3.1.4 the probabilities in R^* that link A to P for a trained classifier on a domain are also considered to be fixed but unknown.

Given the fixed values of the probabilities in R^*, we can consider P_{ij} to be multinomially distributed with $p = r_{ij}$ and $n = a_j$.

$$\mathrm{P} \sim \mathrm{R}^*\mathrm{A}. \tag{3.48}$$

With the test set, the actual class labels are unknown so we only see the total counts by predicted class $\mathbf{p}_t$ and not the full confusion matrix P_t. The relationship between $\mathbf{p}$ and P was given previously in Equation 3.9:

$$\mathbf{p}_t = \mathrm{P}_t \mathbf{1}_n. \tag{3.49}$$

As we are considering P_t to be a matrix of random variables then $\mathbf{p}_t$ is a vector of random variables $\boldsymbol{P}_t$.

So the estimated counts by actual class $\hat{\mathbf{a}}_t$ is also a vector of random variables $\hat{\boldsymbol{A}}_t$ where:

$$\hat{\boldsymbol{A}}_t = \mathrm{R}_v^{-1} \boldsymbol{P}_t. \tag{3.50}$$

3.5.3 General closed-form solution

Temporarily reverting to a 2-class model to simplify the handling of the non-independent variables and to simplify the notation, we can express $\hat{\boldsymbol{A}}_t = \mathrm{R}_v^{-1} \boldsymbol{P}_t$ in a two-class case as:

$$\hat{\boldsymbol{A}}_t = \begin{pmatrix} \hat{A}_{t0} \\ n_t - \hat{A}_{t0} \end{pmatrix} = \begin{pmatrix} R_0 & (1 - R_1) \\ (1 - R_0) & R_1 \end{pmatrix}^{-1} \begin{pmatrix} P_0 \\ n_t - P_0 \end{pmatrix}, \tag{3.51}$$

so:

$$\hat{A}_{t0} = \frac{P_0 - n_t(1 - R_1)}{R_0 - (1 - R_1)}. \tag{3.52}$$

As discussed earlier, we can regard P_0, R_0 and R_1 as normally distributed random variables so we can put $\hat{A}_{t0}$ into the form:

$$\hat{A}_{t0} = \frac{N_1}{N_2}, \tag{3.53}$$

where N_1 and N_2 are themselves normally distributed random variables. If N_1 and N_2 had independent standard[4] normal distributions then the resulting distribution for $\hat{A}_{t0}$ would be a Cauchy distribution [102]. However, R_1 appears in both N_1 and N_2 so we *cannot* regard them as independent. Also N_1 and N_2 will *not* have zero means and therefore will *not* have standard normal distributions. In these circumstances the ratio of N_1 to N_2 becomes considerably more complicated [1] [61]. In my judgement, the level of mathematical complexity to pursue a general solution in closed-form is beyond the scope of a thesis in informatics. Proceeding further with a closed-form solution in this thesis requires simplifying assumptions to be made.

3.5.4 Simplification: n_t is large

The first simplifying assumption is to assume that the test set is large i.e. that n_t is large. If we make this assumption then the Law of Large Numbers can be applied under which we can replace random variables with their mean values. The matrix of random variables R_t can be replaced by the matrix of scalar values R^* meaning that $\boldsymbol{P}_t$ can then simply be regarded as a vector of fixed scalar values $\mathbf{p}_t$.

As the validation set is assumed not to be large (i.e. n_v is not large) then the R_v matrix remains a matrix of random variables.

Referring back to Equation 3.52, the random variable P_0 now becomes a scalar value p_0, but there are no other changes so $\hat{A}_{t0}$ is now:

$$\hat{A}_{t0} = \frac{p_0 - n_t(1 - R_1)}{R_0 - (1 - R_1)}. \tag{3.54}$$

We still have the same problem as in the general case above i.e. while the numerator and

[4] mean of zero and unit standard deviation

denominator are normal random variables they do *not* have independent standard normal distributions. So assuming that n_t is large does *not* significantly help with the closed-form analysis.

3.5.5 Simplification: n_v is large

The second simplifying assumption is to assume that the validation set is large i.e. that n_v is large.

As the test set is *not* large (i.e. n_t is not large), $\boldsymbol{P}_t$ remains a vector of random variables. From Equation 3.50:

$$\hat{\boldsymbol{A}}_t = \mathrm{R}_v^{-1} \boldsymbol{P}_t. \tag{3.55}$$

If we assume that we *do* have a large validation set i.e. that n_v is large then, again, from the Law of Large Numbers:

$$\mathrm{R}_v \to \mathrm{R}^* \tag{3.56}$$

3.5.5.1 With sub-domains (sd)

The estimate of the count by class in the test set $\hat{\boldsymbol{A}}_t$ then becomes:

$$\hat{\boldsymbol{A}}_t = (\mathrm{R}^*)^{-1} \boldsymbol{P}_t. \tag{3.57}$$

To obtain the estimates for class distribution by *main-class* we marginalise out the sub-domains as per Equation 3.43:

$$\hat{\boldsymbol{A}}_{tm} = \begin{pmatrix} \hat{A}_\alpha \\ \hat{A}_\beta \end{pmatrix} = \Theta^T \hat{\boldsymbol{A}}_t = \Theta^T (\mathrm{R}^*)^{-1} \boldsymbol{P}_t. \tag{3.58}$$

As $\hat{A}_\alpha + \hat{A}_\beta = n_t$, the errors in both main class estimates will be the same magnitude but in the opposite direction. As such, we can simply focus on one main-class and we arbitrarily choose class α:

$$\hat{A}_\alpha = \phi^T \Theta^T (\mathrm{R}^*)^{-1} \boldsymbol{P}_t, \tag{3.59}$$

where:

$$\phi = \begin{pmatrix} 1 \\ 0 \end{pmatrix}. \tag{3.60}$$

At this point it is helpful to add two further definitions, firstly Φ:

$$\Phi = \Theta\phi = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \tag{3.61}$$

and secondly $\mathbf{w}$:

$$\mathbf{w} = \left(\Phi^T(\mathrm{R}^*)^{-1}\right)^T, \tag{3.62}$$

so the expression for $\hat{A}_\alpha$ given above in Equation 3.59 now becomes:

$$\hat{A}_\alpha = \mathbf{w}^T \boldsymbol{P}_t, \tag{3.63}$$

where $\mathbf{w}$ is simply a vector of scalar constants:

$$\mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix}. \tag{3.64}$$

So $\hat{A}_\alpha$, the random variable for the estimate of the count by class α in the test set using the sd-method when n_v is large is given by:

$$\hat{A}_\alpha = \mathbf{w}^T \boldsymbol{P}_t = \mathbf{w}^T \mathrm{P}_t \mathbf{1}_4 = \begin{pmatrix} w_1 & w_2 & w_3 & w_4 \end{pmatrix} \begin{pmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \\ P_{41} & P_{42} & P_{43} & P_{44} \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}. \tag{3.65}$$

From this point onward the t sub-scripts are dropped for clarity. By default, absence of a sub-script indicates the test set.

3.5.5.2 Bias: sd

From Equation 3.63 the expected value of $\hat{A}_\alpha$ is given by:

$$\mathbb{E}[\hat{A}_\alpha] = \mathbb{E}[\mathbf{w}^T \boldsymbol{P}] = \mathbf{w}^T \mathbb{E}[\boldsymbol{P}]. \tag{3.66}$$

From the definition of R^* in Section 3.1.4:

$$\mathbb{E}[\boldsymbol{P}] = \mathrm{R}^* \mathbf{a}. \tag{3.67}$$

The mean value of the normal distribution to which we are approximating the distribution of $\hat{A}_\alpha$ is equal to the expected value so:

$$\mu_{\hat{A}_\alpha} = \mathbb{E}[\hat{A}_\alpha] = \mathbf{w}^T \mathrm{R}^* \mathbf{a}. \tag{3.68}$$

The *bias* in the estimate $\hat{A}_\alpha$ is simply:

$$bias(\hat{A}_\alpha) = \mu_{\hat{A}_\alpha} - a_\alpha, \tag{3.69}$$

so:

$$bias(\hat{A}_\alpha) = \mathbf{w}^T \mathrm{R}^* \mathbf{a} - a_\alpha. \tag{3.70}$$

Given that:

$$a_\alpha = \Phi^T \mathbf{a}, \tag{3.71}$$

then:

$$bias(\hat{A}_\alpha) = (\mathbf{w}^T \mathrm{R}^* - \Phi^T)\mathbf{a}. \tag{3.72}$$

Given the expression for $\mathbf{w}$ from Equation 3.62:

$$\mathbf{w}^T = \Phi^T (\mathrm{R}^*)^{-1}, \tag{3.73}$$

then:

$$bias(\hat{A}_\alpha) = (\Phi^T (\mathrm{R}^*)^{-1} \mathrm{R}^* - \Phi^T)\mathbf{a} = 0. \tag{3.74}$$

i.e. under the assumptions in this chapter, when we use sub-domains the estimate is *unbiased*.

3.5.5.3 Variance: sd

Equation 3.65 stated:

$$\hat{A}_\alpha = \mathbf{w}^T \boldsymbol{P}_t = \mathbf{w}^T \mathrm{P}_t \mathbf{1}_4. \tag{3.75}$$

We now define another vector of random variables $\boldsymbol{V}$ as:

$$\boldsymbol{V}^T = \mathbf{w}^T \mathrm{P}, \tag{3.76}$$

so:

$$\hat{A}_\alpha = \boldsymbol{V}^T \mathbf{1}_4 = \sum_{k=1}^{4} V_k. \tag{3.77}$$

Transposing both sides of Equation 3.76 gives:

$$\boldsymbol{V} = \mathrm{P}^T \mathbf{w} \tag{3.78}$$

$$\begin{pmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{pmatrix} = \begin{pmatrix} P_{11} & P_{21} & P_{31} & P_{41} \\ P_{12} & P_{22} & P_{32} & P_{42} \\ P_{13} & P_{23} & P_{33} & P_{43} \\ P_{14} & P_{24} & P_{34} & P_{44} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix}, \tag{3.79}$$

i.e.:

$$V_k = \sum_{j=1}^{4} w_j P_{jk}. \tag{3.80}$$

Each V_k is a random variable which is a weighted sum of the random variables that originally arose from (and sum to) a_k. The four random variables that it is summing are all approximated to the normal distribution so V_k can also be assumed to be an approximation to a normal distribution. However, as the four variables that make up each V_k sum to a_k they are *not* independent of each other.

The variance of V_k is then given by:

$$Var(V_k) = Var\Big(\sum_{j=1}^{4} w_j P_{jk}\Big) = \sum_{i=1}^{4}\sum_{j=1}^{4} w_i w_j Cov(P_{ik}, P_{jk}). \tag{3.81}$$

The variance and covariance of the multinomially distributed random variables (P_{ik}, P_{jk}) is:

$$Cov(P_{ik}, P_{jk}) = -a_k r_{ik} r_{jk}, \tag{3.82}$$

where $i \neq j$ and:

$$Cov(P_{ik}, P_{ik}) = Var(P_{ik}) = a_k r_{ik}(1 - r_{ik}), \tag{3.83}$$

where $i = j$.

If we define the covariance matrix C_k as:

$$\mathrm{C}_k = \begin{pmatrix} c_{11k} & c_{12k} & c_{13k} & c_{14k} \\ c_{21k} & c_{22k} & c_{23k} & c_{24k} \\ c_{31k} & c_{32k} & c_{33k} & c_{34k} \\ c_{41k} & c_{42k} & c_{43k} & c_{44k} \end{pmatrix}, \tag{3.84}$$

where:

$$c_{ijk} = Cov(P_{ik}, P_{jk}), \tag{3.85}$$

then Equation 3.81 becomes:

$$Var(V_k) = \mathbf{w}^T \mathrm{C}_k \mathbf{w}. \tag{3.86}$$

Going back to the equations for covariance (Equation 3.82 and Equation 3.83) we can consider the covariance matrix C_k to be:

$$\mathrm{C}_k = a_k(\mathrm{C}_k^d - \mathrm{C}_k^s), \tag{3.87}$$

where:

$$\mathrm{C}_k^d = \begin{pmatrix} r_{1k} & 0 & 0 & 0 \\ 0 & r_{2k} & 0 & 0 \\ 0 & 0 & r_{3k} & 0 \\ 0 & 0 & 0 & r_{4k} \end{pmatrix}, \tag{3.88}$$

and:

$$\mathrm{C}_k^s = \begin{pmatrix} r_{1k}r_{1k} & r_{1k}r_{2k} & r_{1k}r_{3k} & r_{1k}r_{4k} \\ r_{2k}r_{1k} & r_{2k}r_{2k} & r_{2k}r_{3k} & r_{2k}r_{4k} \\ r_{3k}r_{1k} & r_{3k}r_{2k} & r_{3k}r_{3k} & r_{3k}r_{4k} \\ r_{4k}r_{1k} & r_{4k}r_{2k} & r_{4k}r_{3k} & r_{4k}r_{4k} \end{pmatrix}. \tag{3.89}$$

As each V_k *is* independent of the other random variables V_k then the variance of their sum is simply the sum of their variances so:

$$Var(\hat{A}_\alpha) = \sum_{k=1}^{4} Var(V_k) = \sum_{k=1}^{4} \mathbf{w}^T \mathrm{C}_k \mathbf{w} = \sum_{k=1}^{4} \mathbf{w}^T a_k(\mathrm{C}_k^d - \mathrm{C}_k^s)\mathbf{w} \tag{3.90}$$

$$Var(\hat{A}_\alpha) = \sum_{k=1}^{4} \mathbf{w}^T a_k \mathrm{C}_k^d \mathbf{w} - \sum_{k=1}^{4} \mathbf{w}^T a_k \mathrm{C}_k^s \mathbf{w} \tag{3.91}$$

$$\sum_{k=1}^{4} \mathbf{w}^T a_k \mathrm{C}_k^d \mathbf{w} = \mathbf{w}^T \mathrm{W}\mathrm{R}^* \mathbf{a}, \tag{3.92}$$

where W is the diagonalised matrix of the vector $\mathbf{w}$:

$$\mathrm{W} = \begin{pmatrix} w_1 & 0 & 0 & 0 \\ 0 & w_2 & 0 & 0 \\ 0 & 0 & w_3 & 0 \\ 0 & 0 & 0 & w_4 \end{pmatrix}, \tag{3.93}$$

and:

$$\sum_{k=1}^{4} \mathbf{w}^T a_k \mathrm{C}_k^s \mathbf{w} = \mathbf{w}^T \mathrm{R}^* \mathrm{A} (\mathrm{R}^*)^T \mathbf{w}, \tag{3.94}$$

so:

$$Var(\hat{A}_\alpha) = \mathbf{w}^T \mathrm{W} \mathrm{R}^* \mathbf{a} - \mathbf{w}^T \mathrm{R}^* \mathrm{A} (\mathrm{R}^*)^T \mathbf{w}. \tag{3.95}$$

If again we substitute for $\mathbf{w}^T$ using Equation 3.62 then the expression for $Var(\hat{A}_\alpha)$ when using sub-domains simplifies to:

$$Var(\hat{A}_\alpha) = \Phi^T \big((\mathrm{R}^*)^{-1} \mathrm{W} \mathrm{R}^* - \mathrm{I} \big) \mathbf{a}. \tag{3.96}$$

3.5.5.4 RMSE: sd

As bias is always zero in the sub-domains case (Equation 3.74) the RMSE (Equation 3.24) becomes simply the square-root of the variance:

$$\mathrm{RMSE}_{sd} = \sqrt{\Phi^T \big((\mathrm{R}^*)^{-1} \mathrm{W} \mathrm{R}^* - \mathrm{I} \big) \mathbf{a}}. \tag{3.97}$$

3.5.5.5 Without sub-domains (nsd)

The approach taken here is to get the estimate $\hat{A}_\alpha$ into the same form as Equation 3.63 i.e:

$$\hat{A}_{n\alpha} = \mathbf{w}_n^T \boldsymbol{P}_t. \tag{3.98}$$

The subscript n designating that these variables relate to the nsd-method.

Finding the mean, variance and RMSE error is then simply a matter of substituting values of $\mathbf{w}_n$ into in the formulae in the previous section.

From Equation 3.39:

$$\mathrm{R}_{nv} = \Theta^T \mathrm{P}_v \Theta (\Theta^T \mathrm{A}_v \Theta)^{-1}. \tag{3.99}$$

As n_v is large then by The Law of Large Numbers:

$$\mathrm{P}_v = \mathrm{R}^* \mathrm{A}_v, \tag{3.100}$$

so:

$$\mathrm{R}_{nv} = \Theta^T \mathrm{R}^* \mathrm{A}_v \Theta (\Theta^T \mathrm{A}_v \Theta)^{-1}. \tag{3.101}$$

Given also Equation 3.40

$$\mathbf{p}_{nt} = \Theta^T \mathbf{p}_t, \tag{3.102}$$

and Equation 3.41

$$\hat{\mathbf{a}}_{nt} = \mathrm{R}_{nv}^{-1} \mathbf{p}_{nt}, \tag{3.103}$$

and considering that $\mathbf{p}_t$ and $\hat{\mathbf{a}}_{nt}$ are now vectors of random variables $\boldsymbol{P}_t$ and $\hat{\boldsymbol{A}}_{nt}$ then:

$$\hat{\boldsymbol{A}}_{nt} = \left(\Theta^T \mathrm{R}^* \mathrm{A}_v \Theta (\Theta^T \mathrm{A}_v \Theta)^{-1}\right)^{-1} \Theta^T \boldsymbol{P}_t \tag{3.104}$$

$$\hat{\boldsymbol{A}}_{nt} = \Theta^T \mathrm{A}_v \Theta (\Theta^T \mathrm{R}^* \mathrm{A}_v \Theta)^{-1} \Theta^T \boldsymbol{P}_t, \tag{3.105}$$

so:

$$\hat{A}_{n\alpha} = \phi^T \Theta^T \mathrm{A}_v \Theta (\Theta^T \mathrm{R}^* \mathrm{A}_v \Theta)^{-1} \Theta^T \boldsymbol{P}_t, \tag{3.106}$$

which is now in the same form as Equation 3.98:

$$\hat{A}_{n\alpha} = \mathbf{w}_n^T \boldsymbol{P}_t, \tag{3.107}$$

i.e.:

$$\mathbf{w}_n^T = \phi^T \Theta^T \mathrm{A}_v \Theta (\Theta^T \mathrm{R}^* \mathrm{A}_v \Theta)^{-1} \Theta^T. \tag{3.108}$$

We can now use $\mathbf{w}_n$ in the previously derived equations for bias and variance.

3.5.5.6 Bias: nsd

Using Equation 3.68 to get the mean of $\hat{A}_{n\alpha}$ gives:

$$\mu_{\hat{A}_{n\alpha}} = \mathbf{w}_n^T \mathrm{R}^* \mathbf{a}. \tag{3.109}$$

From Equation 3.72 we get an expression for the bias of the distribution of $\hat{A}_{n\alpha}$:

$$bias(\hat{A}_{n\alpha}) = (\mathbf{w}_n^T \mathrm{R}^* - \Phi^T)\mathbf{a}. \tag{3.110}$$

3.5.5.7 Variance: nsd

From Equation 3.95 we get an expression for the variance of the distribution of $\hat{A}_{n\alpha}$:

$$Var(\hat{A}_{n\alpha}) = \mathbf{w}_n^T\big(\mathrm{W}_n\mathrm{R}^*\mathbf{a} - \mathrm{R}^*\mathrm{A}(\mathrm{R}^*)^T\mathbf{w}_n\big). \tag{3.111}$$

3.5.5.8 RMSE: nsd

So the Root Mean Squared Error (Equation 3.24) when sub-domains are ignored and n_v is large is given by:

$$\mathrm{RMSE}_{nsd} = \sqrt{\big((\mathbf{w}_n^T\mathrm{R}^* - \Phi^T)\mathbf{a}\big)^2 + \mathbf{w}_n^T\big(\mathrm{W}_n\mathrm{R}^*\mathbf{a} - \mathrm{R}^*\mathrm{A}(\mathrm{R}^*)^T\mathbf{w}_n\big)}. \tag{3.112}$$

3.5.5.9 ΔRMSE

Defining ΔRMSE as:

$$\Delta\mathrm{RMSE} = \mathrm{RMSE}_{nsd} - \mathrm{RMSE}_{sd}, \tag{3.113}$$

from Equation 3.97 and Equation 3.112 we now have:

$$\Delta\mathrm{RMSE} = \sqrt{\big((\mathbf{w}_n^T\mathrm{R}^* - \Phi^T)\mathbf{a}\big)^2 + \mathbf{w}_n^T\big(\mathrm{W}_n\mathrm{R}^*\mathbf{a} - \mathrm{R}^*\mathrm{A}(\mathrm{R}^*)^T\mathbf{w}_n\big)} - \sqrt{\Phi^T\big((\mathrm{R}^*)^{-1}\mathrm{W}\mathrm{R}^* - \mathrm{I}\big)\mathbf{a}}. \tag{3.114}$$

Ideally we would like a usable closed form expression for ΔRMSE that would allow us to know when using sub-domains gives smaller errors and when it gives larger errors. Unfortunately, the above expression for ΔRMSE does not simplify to a readily usable expression. The expression may be correct (see Section 3.6) but it is not particularly *useful*.

Attempts to simplify the expression through parameterisation of its terms did not result an expressions that were simpler or more usable. For an assumed R^* (or from observation given that for a large n_v: $\mathrm{R}^* \approx \mathrm{R}_v$) and counts by class $\mathbf{a}$, the expected value of RMSE

can be calculated using Equation 3.114. However, it may be as simple to just do this with numerical simulation.

3.6 Numerical validation of the closed-form solution

This aim of this section is use numerical simulation to validate the closed-form expressions that were derived for both the sd and nsd methods in the Section 3.5.

3.6.1 Method

Validation and test sets were repeatedly synthesised based on set parameters and in accordance with the assumptions as set out at the start of this chapter, most importantly that recall is treated as constant within each main-class/sub-domain combination. Quantification both with and without using sub-domains was then carried out using those datasets.

If the formulae are correct then the difference between the error computed from the formulae and the error computed from the simulated data itself will reduce to zero as the size of the validation set n_v increases.

Data: Synthetic

Result: Formula and empirical values

while *Outer Loop count* $< 20,000$ **do**

  Sample the recall parameters;

  Sample the validation set size;

  Sample the test set size;

  Sample the test set main-class and sub-domain proportions;

  Construct computed variables: R^*, $\mathbf{a}_t$ etc.;

  while *Inner Loop count* $< 5,000$ **do**

    Generate P_v from $\mathbf{a}_v$ and the probabilities in R^*;

    Generate $\mathbf{p}_t$ from $\mathbf{a}_t$ and the probabilities in R^*;

    Compute $\hat{A}_\alpha$ with sd-method;

    Compute $\hat{A}_{n\alpha}$ with nsd-method;

  end

  Compute empirical values from inner-loop results;

  Compute formula values from outer-loop parameters;

end

Algorithm 2: Numerical validation of closed-loop formulae

As the size of the test set varied in the simulation, RMSE was normalised by dividing by the size of the test set (n_t).

3.6.2 Results

Figure 3.9 shows the difference in RMSE between the observed values from the simulation and the value computed using Equations 3.97 and 3.112, against the size of the validation set.

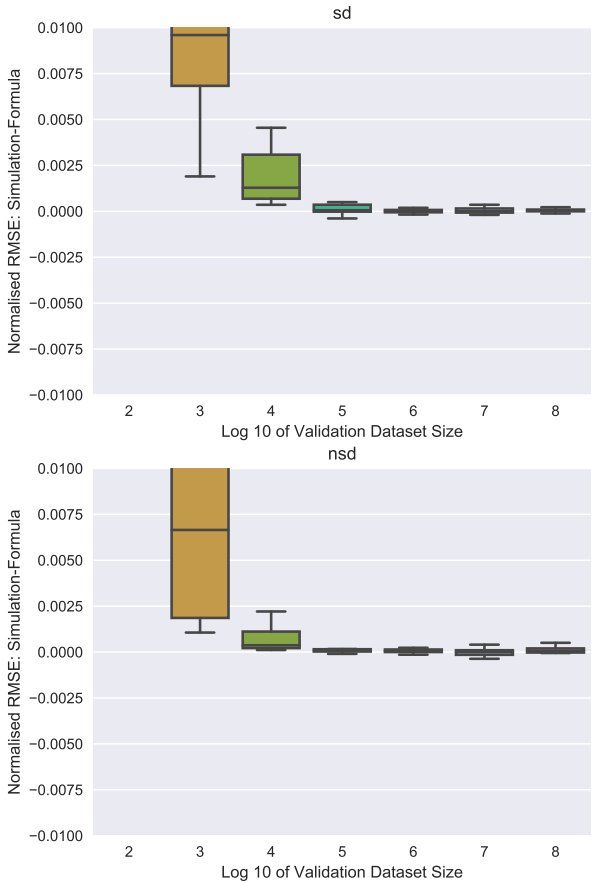

Figure 3.9: Normalised RMSE of estimate for size of main-class α: simulation values less values from Equations 3.97 and 3.112 vs. log10 of size of validation set n_v

As the size of the validation set n_v becomes larger, the value of RMSE calculated from Equations 3.97 and 3.112 converges with the observed values.

Similar results were found separately for both bias and variance. In all cases the value from the numerical simulation converged to the value from the formula as the size of the validation set n_v increased. It appears from that the closed-form solutions in Section 3.5 are valid.

3.7 Quantification accuracy and classifier accuracy

The principle behind *classify and adjust* methods is that the estimate of class proportions from counting the classifier outputs by class is adjusted to effectively negate the effect of the classifier inaccuracy.

However, with finite-sized test and validation sets, stochasticity has a double impact on quantification accuracy.

Firstly on the estimation of the true values of classifier recall, R^*. We use a finite amount of labelled validation data to compute R_v which is our estimate of R^*. Equation 3.16 gives

$$\mathrm{R}_v = \mathrm{P}_v(\mathrm{A}_v)^{-1}. \tag{3.115}$$

While A_v is fixed, the values in P_v are stochastic with its values distributed multi-nomially. Restating Equation 3.48:

$$\mathrm{P}_v \sim \mathrm{R}^*\mathrm{A}_v. \tag{3.116}$$

Secondly the counts by predicted class for the test set will also be stochastic and distributed multi-nomially:

$$\mathbf{p}_t \sim \mathrm{R}^*\mathbf{a}_t. \tag{3.117}$$

The the estimate of class membership, $\hat{\mathbf{a}}_t$ is given by Equation 3.29:

$$\hat{\mathbf{a}}_t = \mathrm{R}_v^{-1}\mathbf{p}_t. \tag{3.118}$$

So when the estimate of class membership, $\hat{\mathbf{a}}_t$, is computed the two stochastic effects are compounded. Figure 3.10 shows results from a simulation of a matrix-inversion *classify and adjust* quantifier with varying validation and test set sizes (n) and classifier recall values.

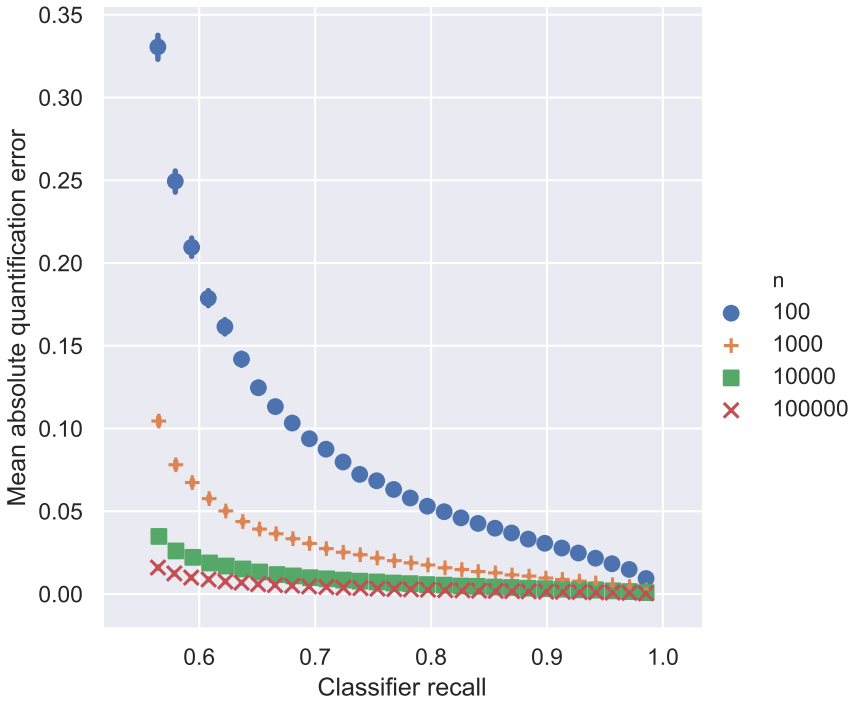

Figure 3.10: Mean absolute quantification error using classify and adjust method vs. classifier recall and dataset size(n). Simulated data.

Higher levels of classifier recall lead to lower quantification errors. A perfect classifier is a perfect quantifier. However, Figure 3.10 indicates that lower levels of recall may not be a practical issue if the validation and test sets are sufficiently large.

3.8 Exploration of explicit sub-domains through simulation

In Section 3.5, closed-form expressions were derived for the expected quantification error when using, and when not using, explicit sub-domains. These were validated in Section 3.6. It was only possible to derive expressions for the case when the validation set was large and the resulting expressions do not readily simplify. In this section, numerical simulation is used to explore the the impact of individual parameters.

3.8.1 Method

The work in this section used the same basic program code that was used for numerical validation of the closed-form formulae that was outlined in Section 3.6.1.

3.8.2 Simulation settings

The numerical simulation was driven by a number of parameters shown in Tables 3.8 and 3.9.

Table 3.8: Numerical simulation: fixed parameter values

Parameter	Value
Outer loops	10,000
Inner loops	200
Recall parameters[5]	8
Class-balanced validation set	True

Table 3.9: Numerical simulation: sampled parameter values

Parameter	Lower	Upper	Distribution
Main-class recall target	0.6	0.95	Uniform
Sub-domain recall target	0.6	0.95	Uniform
Validation set size n_v	100	10,000,000	Uniform $\log_{10}$
Test set size n_t	100	10,000,000	Uniform $\log_{10}$
Test set main-class proportion	0.05	0.95	Uniform
Test set sub-domain proportion	0.05	0.95	Uniform

A *target* recall value for each main-class was sampled from a uniform distribution. This value was then used as the mode for a beta distribution from which the main-class recall values were sampled for each sub-domain. This gave a correlation between the recall values as shown in Figure 3.11 below.

[5]The R^* matrix is constructed from main-class and sub-domain recall parameters

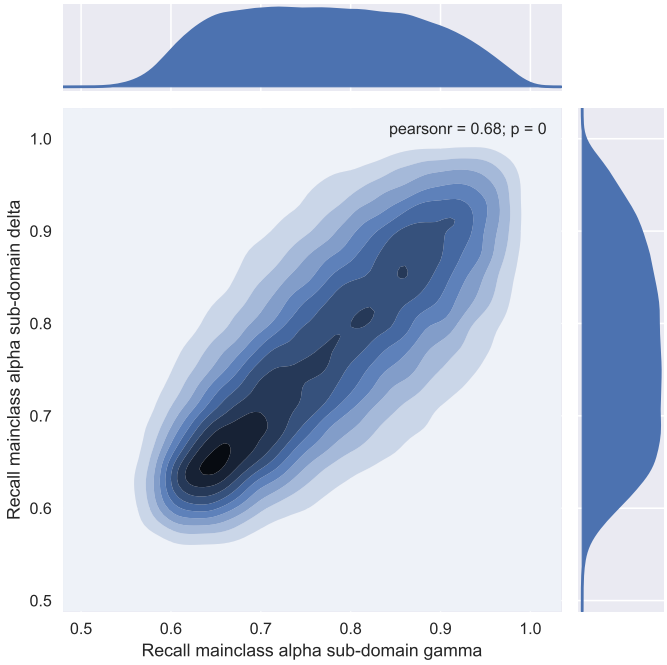

Figure 3.11: Kernel density estimation plot showing correlation of main-class α recall values between the two sub-domains γ and δ

3.8.3 Validation set size n_v

Figure 3.12 shows the mean quantification error (RMSE) against the size of the validation dataset for the sd and nsd methods separately.

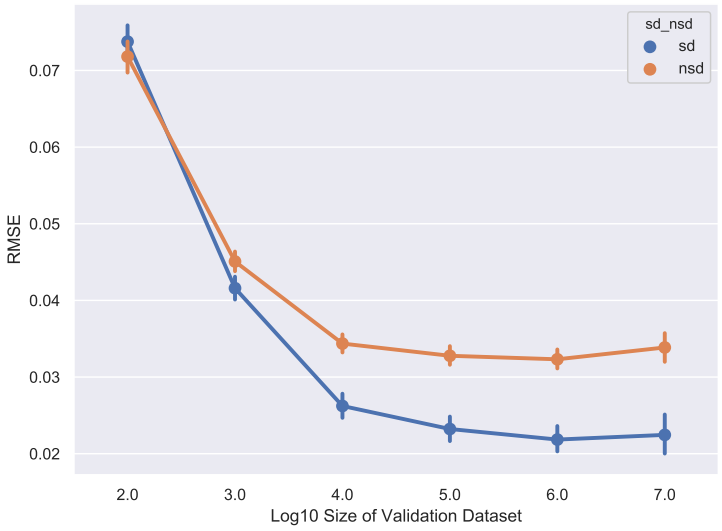

Figure 3.12: Quantification error in RMSE for the sd and nsd methods vs. log_{10} of validation set size

It shows that while the accuracy of both methods improves as the validation set increases in size, the sd method is more accurate than the nsd method when the validation set is large but can be worse then the nsd method when the validation set is small. This difference in performance is shown with the same data in Figure 3.13.

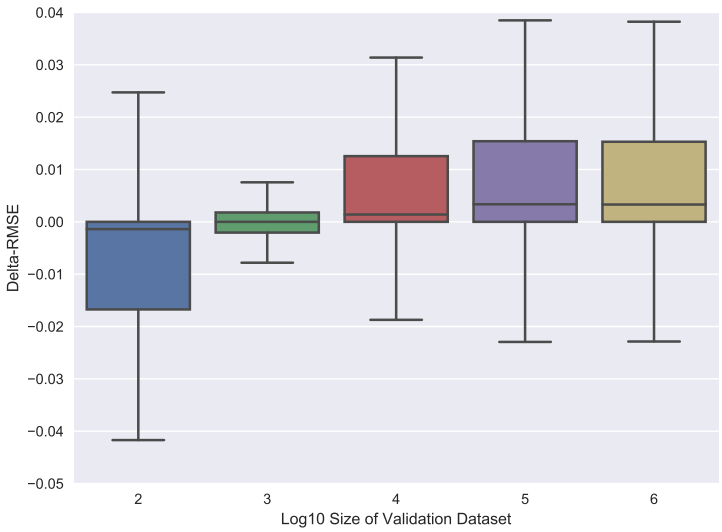

Figure 3.13: Boxplot[6]of ΔRMSE against size of validation set

The under-performance of the sub-domain method when the amount of validation data is small is effectively a noise problem. Without sub-domains, the validation data is used to compute an R matrix containing 4 recall values. With sub-domains it is used to calculate a matrix with 16. With one quarter the amount of data per 'class', the impact of binomial noise is much more significant.

To show the effect of this noise the simulation was re-run with the underlying R* values of main-class recall set to be the *same* for both sub-domains (see Figure 3.14). Any difference in observed main-class recall by sub-domain is then purely as a result of noise in the samples.

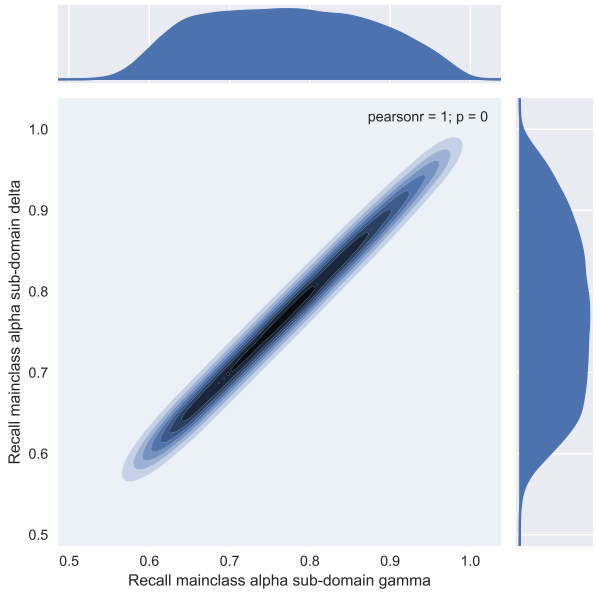

Figure 3.14: Kernel density estimation plot showing correlation of main-class α recall values between the two sub-domains γ and δ

Figure 3.15 shows that, as expected, the effect of multi-nomial noise had a large negative effect on quantification error for the sd-method relative to the nsd-method when the amount of available validation data was small.

[6]Seaborn Boxplot documentation: "The box shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution, except for points that are determined to be "outliers" using a method that is a function of the inter-quartile range."

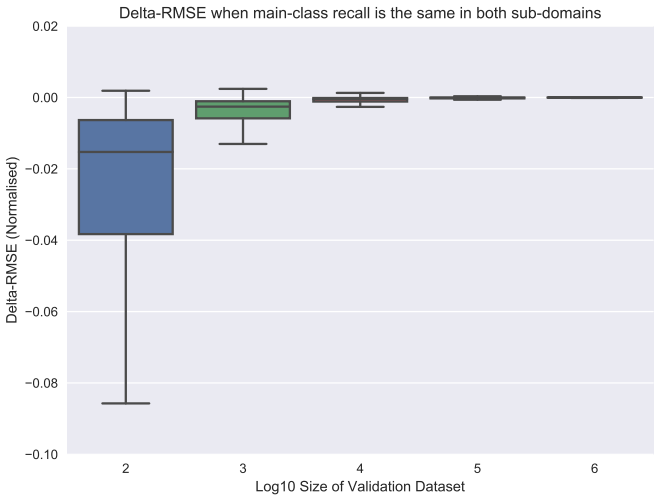

Figure 3.15: Boxplot ΔRMSE against size of validation set when main-class recall is the same in both sub-domains

3.8.4 Multiple regression analysis

The candidate independent variables and the dependent variable were all standardised to zero mean and unit standard deviation. The independent variables consisted of both the original parameters and their logarithms. The dependent variable was ΔRMSE. Ordinary Least Squares (OLS) regression was performed multiple times, each time dropping the variable that contributed least as judged by its confidence interval. In this way the initial set of 12 independent variables was reduced to the 6 shown in Table 3.10.

Table 3.10: Numerical simulation: results of OLS regression using 6 parameters, full range of parameter values

Parameter	Coefficient	2.5% CI	97.5% CI
Test set size (log10)	0.154	0.146	0.162
Validation set size (log10)	0.423	0.415	0.431
Main-class recall mean	0.034	0.026	0.042
Main-class recall difference by sd[7]	0.283	0.275	0.291
Sub-domain recall mean	0.093	0.085	0.101
Sub-domain proportion difference[8]	0.203	0.195	0.211
R-squared	0.334		
Number of observations	40,000		

The R-squared value of 0.334 is quite low indicating that, as would be expected, a simple linear model is not a good fit for the actual observed difference in errors.

The largest coefficient is for *Validation set size (log10)*. This is not unexpected given the observations made in Section 3.8.3 earlier. To reduce the impact of the validation set size and explore other coefficients, the regression was re-run after removing the observations where n_v was less than 10,000. The results are given below in Table 3.11.

Table 3.11: Numerical simulation: results of OLS regression using 6 parameters, $n_v > 10,000$

Parameter	Coefficient	2.5% CI	97.5% CI
Test set size (log10)	0.250	0.240	0.260
Validation set size (log10)	0.054	0.045	0.064
Main-class recall mean	-0.120	-0.130	-0.110
Main-class recall difference by sd	0.459	0.449	0.469
Sub-domain recall mean	0.024	0.015	0.034
Sub-domain proportion difference	0.327	0.317	0.337
R-squared	0.384		
Number of observations	24,174		

[7]The absolute difference within each main-class weighted by the size of the main-class

[8]The absolute difference between the proportion of sub-domain gamma and sub-domain delta

The coefficient for *Validation set size (log10)* has dropped very considerably. This is as previously observed in Figure 3.13. Once the validation set size is above a certain size the impact of further size increases is much smaller.

The impact of other individual parameters is discussed below.

3.8.5 Main-class recall

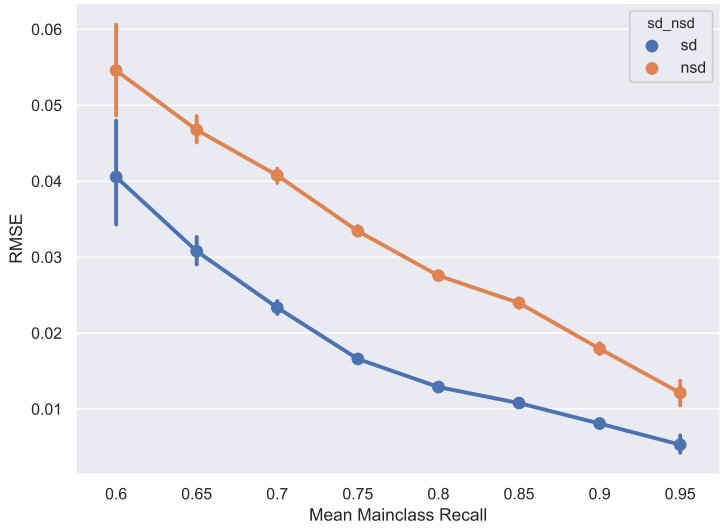

Figure 3.16: RMSE vs. main mainclass recall. 95% CI shown.

The results of the multiple regression analysis in Table 3.11 show a negative correlation with *Main-class recall mean*. This is initially perplexing, that an increase in main-class recall favours the method that does not use sub-domains over the method that does. Figure 3.16 shows how quantification error measured by RMSE varies with main-class recall for both the sd-method and the nsd-method. It appears that this negative correlation arises because RMSE is initially higher and decreases more quickly in the nsd method than with the sd method.

3.8.6 Sub-domain recall

A reasonable hypothesis would be that for the sub-domains method to be effective it requires that sub-domains can be accurately classified i.e. that there is a positive correlation between sub-domain recall and ΔRMSE.

However, the results of the simulation in Figure 3.17 do not show a strong correlation:

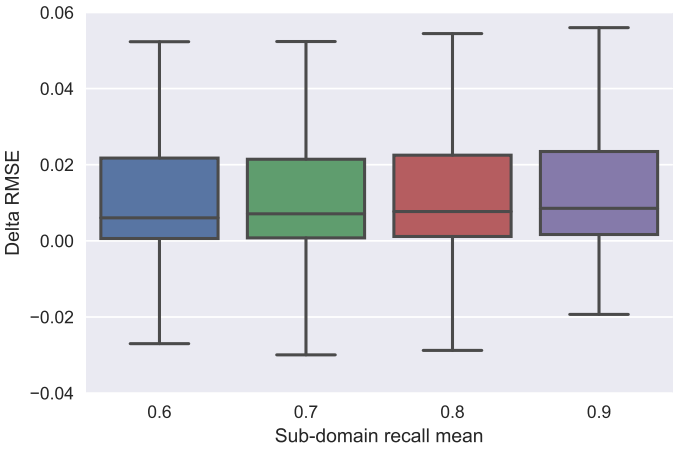

Figure 3.17: ΔRMSE vs. sub-domain recall mean

A simple t-test on the ΔRMSE values when $r_s = 0.6$ vs. when $r_s = 0.9$ gave a p-value of 0.0001 indicating that it is unlikely that there is no difference. However any difference is small relative to the differences seen with some other parameters.

3.8.7 Test set size

As seen with the OLS multivariate regression in Section 3.8.4 above, the sd-method outperforms the nsd-method as the test set becomes large.

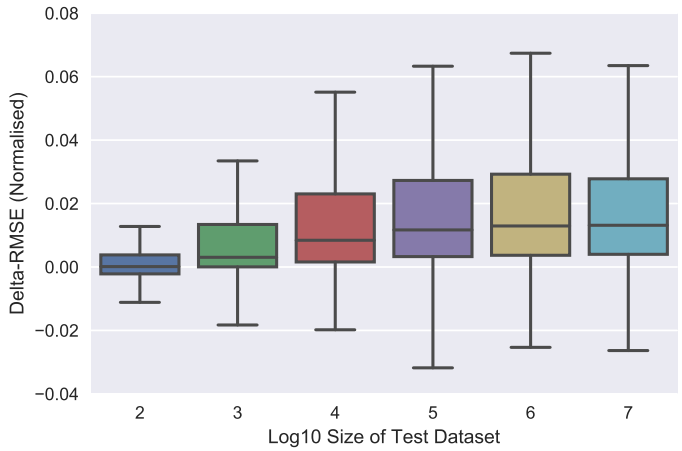

Figure 3.18: ΔRMSE vs. Log10 of size of test set when validation set >10,000

Again, the hypothesis is that this relates to noise: that as the size of the test set increases

the variance due to mis-classification reduces and the effect of the actual difference in recall between sub-domains can be seen.

3.9 Conclusions

Initial experiments with Twitter data showed that introducing explicit sub-domains in the matrix-inversion method (the *sd-method*) reduced *bias* but increased *variance*. The intuition was that it might be possible to derive a closed-form expression for expected quantification error for quantification using matrix-inversion and explicit sub-domains. This was possible for the case when there was a large amount of labelled data available. A closed-form expression for the general case may also be possible, but deriving it is likely to be very complex and its usefulness is likely to be limited.

The limitations of the closed-form approach justified the use of numerical simulation for further exploration. The numerical simulation showed a strong relationship between the relative performance of the sd and nsd methods and the size of the validation set n_v. Under the parameters of the simulated dataset when n_v was small the nsd-method was more accurate while when n_v was large the sd-method was better. In the simulation the transition between the two was when n_v was around 1000.

The simulation also showed that the advantage of the sd-method over the nsd-method was correlated to other parameters, most strongly to the difference in main-class recall between sub-domains and to the difference in the proportions of the sub-domains (relative the validation set where the proportions were balanced). The size of the test set was correlated while sub-domain recall, the ability of the classifier to assign instances to the correct sub-domain, was found to be much less strongly correlated.

However, it is still not clear at this stage how to specifically determine in advance whether the sd or the nsd method will give the highest quantification accuracy. This is the focus for Chapter 4, whether the insights from this chapter about the relative performance of the two methods can be used in a method that gives a meaningful and reliable improvement on the baseline nsd method.

Chapter 4

Domain adaptation with thresholded sub-domains

The work in Chapter 3 indicated that a method using explicit sub-domains *can* improve quantification accuracy. However, it also showed that using explicit sub-domains can also *reduce* quantification accuracy. Whether a method using explicit sub-domains gives better quantification accuracy than one which does not was shown to be correlated to a number of parameters.

The aim of this chapter is to see whether an effective method, the *Thresholded Sub-Domains* (*tsd*) method, can be devised that uses explicit sub-domains when the value of certain parameters indicates that it will increase quantification accuracy.

A simple approach would be to use a threshold based on validation set size. A clear relationship between the effectiveness of using explicit sub-domains and validation set size was shown with simulated data in Chapter 3. Explicit sub-domains could be used only when the validation dataset was sufficiently large. However, it may be be difficult to define a value for 'sufficiently large' that works for all potential domains that may be encountered.

An alternative, and perhaps more principled approach, would be to use the statistical significance of the observed difference in main-class recall between the explicit sub-domains. The results in Chapter 3 showed that differences in recall due *solely* to random sampling could have a large negative impact on quantification accuracy when using explicit sub-domains. Using explicit sub-domains *only* when the difference in recall was unlikely to be

due to such a random effect could be a robust solution.

The problem is explored with data from three sources: with *simulated data* in Section 4.1, with *public-domain datasets* from the UCI repository in Sections 4.2 and 4.3 and with a dataset of *Twitter users* in Section 4.4.

4.1 Experiment 1: Simulation

In Chapter 3, simulated data was used to explore the performance of a quantification method using explicit sub-domains. This section takes a similar approach. Simulated data is used to explore whether criteria based on thresholds can determine whether quantification using explicit sub-domains will give a better performance than quantification that does not use explicit sub-domains.

4.1.1 Method

In each iteration, a validation set and a biased test set was simulated, with the simulation being controlled by a range of fixed and variable parameters. In the validation sets the main-class and sub-domain proportions remained fixed and balanced. In the test sets the main-class and sub-domain proportions varied. For each validation and test set pairing the main-class proportions in the test set were estimated using the matrix-inversion method seen in Chapter 3. In each case the estimate of main-class proportions was made both using explicit sub-domains (sd-method) and not using explicit sub-domains (nsd-method).

Bootstrapping was used to estimate the probability that the difference in main-class recall between the two sub-domains was due to a genuine difference in underlying recall probabilities rather than arising simply as a product of random sampling. The null-hypothesis was that there is *no difference* in underlying main-class recall between the two sub-domains. Using the validation set, recall values were calculated for each main-class in total, ignoring sub-domains. These recall values were then used as the probability of correctly classifying and instance from each class. Bootstrap samples were constructed using those probabilities and the actual size of each main-class and sub-domain in the validation set. *Observed* recall values were then computed from each bootstrap set. The p-value for each main-class was then computed as the proportion of the bootstrap samples where the difference in observed recall values between the sub-domains was greater than that seen in the original

validation set.

Data: Synthetic

Result: Estimated test set class proportions by nsd and sd-methods and bootstrap null-hypothesis proportion

while *Loop count* $< 100,000$ **do**

Sample the recall parameters, validation set size etc.;

Construct computed variables: R*, $\mathbf{a}_t$ etc.;

Generate P_v by random sampling using probabilities from R^* and values from A_v;

Generate R_v from P_v and A_v;

Marginalise R_v into 8 recall values and construct R_{v8} from these values;

Marginalise R_{v8} into R_{vm0} by assuming that there is no difference in main-class recall between sub-domains;

Calculate A: = Difference in main-class recall between sub-domains in R_{v8}

while *Bootstrap count* $< 10,000$ **do**

Generate P_{vb} by random sampling using probabilities from R_{vm0} and values from A_v;

Generate R_{vb} from P_{vb} and A_v;

Calculate B = Difference in main-class recall between sub-domains in R_{vb};

if $B > A$ **then**

add 1 to null-hypothesis counter

end

end

Compute bootstrap null-hypothesis proportions from null-hypothesis counts and number of bootstraps;

Compute $\hat{\mathbf{a}}_t$ using both sd and nsd-methods;

Write parameters and results to file;

end

Algorithm 3: Experiment 1

As is normal with the matrix-inversion method, the values for $\hat{\mathbf{a}}_t$ were clipped to lie between zero and the size of the test set.

4.1.2 Simulation settings

The parameter values are shown in Tables 4.1 and 4.2.

Table 4.1: Numerical simulation: fixed parameter values

Parameter	Value
Iterations (Outer-Loops)	100,000
Bootstraps (Inner-Loops)	10,000
Recall parameters	8
Test set size n_t	1,000
Class-balanced validation dataset	True

Sub-domain was labelled for all data, including the test data, and did not have to be estimated using a classifier as was the case in previous chapters.

Table 4.2: Numerical simulation: sampled parameter values

Parameter	Lower	Upper	Distribution
Main-class recall target	0.6	0.95	Uniform
Validation set size n_v	100	100,000	7 discrete values
test set main-class proportion	0.05	0.95	Uniform
test set sub-domain proportion	0.05	0.95	Uniform

4.1.3 Results

4.1.3.1 Validation dataset size

Figure 4.1 shows the difference in Mean Absolute Error (MAE) between the sd and the nsd-methods against the validation dataset size (n_v).

Figure 4.1: Mean delta absolute error nsd-method minus absolute error sd-method by validation dataset size. 95% confidence intervals shown.

The findings are consistent with the earlier observation seen in Section 3.8.3 that when the validation dataset is below a certain size the quantification accuracy of the sd-method drops below that of the nsd-method.

4.1.3.2 Statistical significance of sub-domain recall difference

The bootstrap method generates separate p-values for the two main-classes but a single combined value is useful for analysis. In quantification the class balance of the test set is, of course, unknown so the joint probability[1] was selected as the single metric (*bs_prop_prod*) to ensure that the p-value is low for *both* main-classes. A $\max(p_0, p_1)$ function would have been another option.

Figure 4.2 below show the difference in Mean Absolute Error (MAE) between the sd and the nsd-method against bs_prop_prod quartiles.

[1] The product of the p-values for the two classes

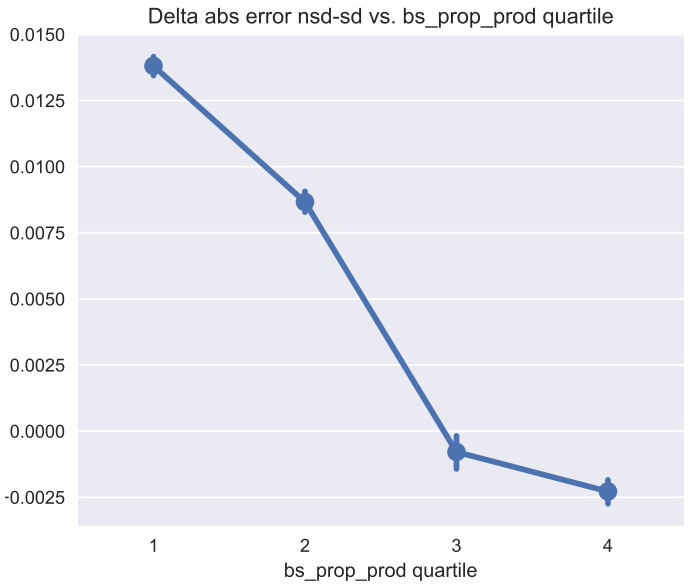

Figure 4.2: Mean delta absolute error (nsd-method minus absolute error sd-method) by quartile of bs_prop_prod. 95% confidence intervals shown.

When the value of bs_prop_prod is low (i.e. when the difference in main-class recall values between sub-domains is most significant) the sd-method typically gives a lower error than the nsd-method. In this experiment, for the first quartile of bs_prop_prod values, the sd-method has a mean absolute error of around 1.4 percentage points lower than the nsd-method.

4.1.3.3 Statistical significance of difference in recall by subdomain *and* difference in sub-domain proportions

When there is no difference in the distribution of sub-domains within each main-class between the test set and the validation set[2] then, in this experimental setup, there will be no difference in overall main-class recall between the validation and test sets. If this is the case then clearly, using sub-domains for quantification will not give any advantage over the nsd-method.

I defined the metric of *Sub-Domain Distance (SDD)* to capture the difference in sub-domain distribution within the main-classes as a single value. It is modelled on Euclidean

[2] i.e. we see no class-conditional dataset shift

Distance. Importantly, it is independent of the difference in main-class proportions between the validation and test sets.

$$SDD = \sqrt{\left(\frac{a_{v\alpha\gamma}}{a_{v\alpha}} - \frac{a_{t\alpha\gamma}}{a_{t\alpha}}\right)^2 + \left(\frac{a_{v\beta\gamma}}{a_{v\beta}} - \frac{a_{t\beta\gamma}}{a_{t\beta}}\right)^2 + \left(\frac{a_{v\alpha\delta}}{a_{v\alpha}} - \frac{a_{t\alpha\delta}}{a_{t\alpha}}\right)^2 + \left(\frac{a_{v\beta\delta}}{a_{v\beta}} - \frac{a_{t\beta\delta}}{a_{t\beta}}\right)^2} \tag{4.1}$$

As defined in Section 3.1.3, the main-classes and are designated as α and β while the two sub-domains are designated as γ and δ. A large value of SDD indicates a large difference in sub-domain proportions within each main-class between the validation and the test set.

Figure 4.3 is a heatmap showing the difference in RMSE between the nsd and sd-methods by SDD quartile and by bs_prop_prod quartile.

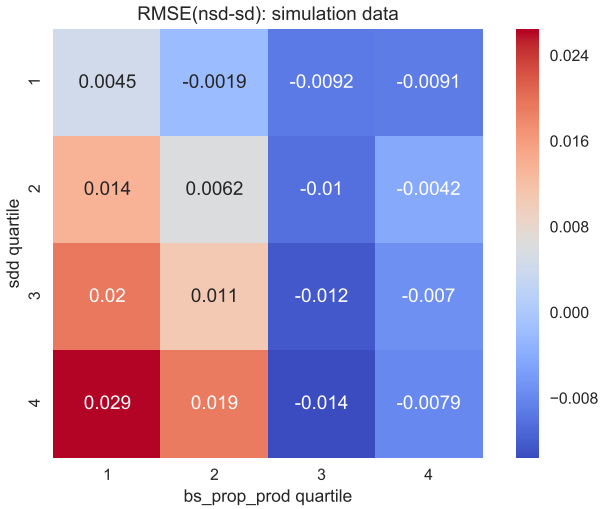

Figure 4.3: Delta RMSE (nsd-sd) by quartile of bs_prop_prod and quartile of SDD

As expected, when bs_prop_prod is low[3] (1st quartile) the advantage of the sd-method over the nsd-method is clearly dependent on the difference in sub-domain proportions, in this case as measured using the SDD metric. When bs_prop_prod is low (quartile 1) and SDD is high (quartile 4) then the RMSE of the sd-method is 2.9 percentage points lower than the RMSE of the nsd-method.

[3]i.e. the main-class recall difference is unlikely to have arisen purely by random chance

4.1.4 Discussion

On its own, high statistical significance (as given by low values of bs_prop_prod) does identify situations where carrying out quantification using sub-domains can give a better quantification accuracy. Unfortunately, to avoid situations where quantification accuracy could actually be reduced the threshold would have to be set at a level where only 25% of the results would be selected.

4.2 Experiment 2: UCI datasets

The objective of this second experiment was to see if the observations made on simulated data in Section 4.1 would be replicated on real datasets.

4.2.1 Datasets

I downloaded seven datasets from the University of California at Irvine (UCI) repository [32]. The original experiment with a Twitter dataset in Section 3.4 had used a dataset with very high dimensionality so for consistency UCI datasets were chosen which also had higher dimensionality. The other main selection criteria was that the dataset should be sufficiently large to allow multiple samples to be drawn of varying sizes. The seven datasets represent a wide variety of domains, from census data on individuals to the chemical properties of proteins.

Any real values in the datasets were standardised to zero mean and unit standard deviation and any categorical values were converted to multiple binary features with one-hot encoding. Real or categorical labels were converted to binary labels with a logical cut-off or grouping that would result in a reasonably balanced class distribution.

Finally, the datasets were arbitrarily split into two groups: 4 for development purposes, shown in Table 4.3, and 3 which were held-out for final testing shown in Table 4.4:

Table 4.3: Development datasets: UCI_dev

Dataset	Description	Number of Instances n	Number of Features d
Adult	Census data from 1994 Label: person earns >$50k	15,061	56
Bank Marketing	Customer data from a marketing campaign Label: person took up offer	45,211	34
Covertype	Predicting forest cover type from cartographic variables Label: categorical - type of cover	581,012	54
Letter Recognition	Integer values computed from images of letters Label: letter of alphabet	20,000	16

Table 4.4: Held-out test datasets: UCI_test

Dataset	Description	Number of Instances n	Number of Features d
Casp	Physicochemical properties of protein tertiary structure Label: size of the residue	45,730	9
Credit Card Default	Customer personal data and financial history Label: defaulted on payment	25,318	42
Online News Popularity	Statistics extracted from online news articles Label: number of shares	39,644	58

4.2.2 Generation of results

In each iteration a train-validation ('trv') set and a test set was sampled from the dataset. The trv set was class and sub-domain balanced. The test set was built to a given class balance and given sub-domain balance. On each iteration, one of the binary features in the dataset was randomly selected to indicate the sub-domain. This could be one of the original features or one of the 'binarised'[4] versions of the first five Principle Component Analysis (PCA) components. The recall values by class and by sub-domain were computed with 5-times cross validation on the trv set, then the classifier was trained with the full trv set.

While the original features might represent real-world sub-domains such as age, gender or occupation status in UCI Adult, the PCA features are clearly abstractions and may not map onto any any real-world attribute.

In this experiment the sub-domain is explicit and known in *both* the trv *and* test sets. It was not necessary to estimate the sub-domain label in the test set. As such it was not necessary to treat this as a 4-class problem as it was in Section 3.3. Instead it was treated as a set of 2-class quantification problems. 2-class quantification was carried out both separately for each sub-domain and then aggregated (the sd-method) and also for all the data together without consideration of sub-domains (the nsd-method).

In Section 4.1 bootstrapping was used to generate a p-value. In this experiment the p-value was simply computed using Pearson's chi-squared test. Again, this gives us the likelihood of seeing at least the number of correctly and incorrectly classified instances within each sub-domain if the null-hypothesis were true that there was no difference in classifier recall between the two sub-domains[5].

As with bootstrapping in Section 4.1, the chi-squared test gives a separate p-value for mainclass 0 and for mainclass 1 where it would be useful to have a single metric for analysis purposes. Again, the joint probability was used. The final metric *log10_c2p_sum* was the negative $\log_{10}$ of the product of the two p-values.

[4] Assigned a value of 1 if feature value is greater than zero and 0 otherwise

[5] Not a strictly accurate definition but one which will suffice for these purposes

4.2.3 Algorithm

Data: 7 UCI Datasets

Result: Estimated test set class proportions by nsd and sd-methods

for *dataset in set of UCI Datasets* **do**

  import the dataset X and y;

  for *feature in features suitable for use as sub-domain* **do**

    use the binary value of that feature to indicate sub-domain;

    for *log10(trv size) in* [2.0 *to* 4.0] **do**

      find optimal C value using 4-fold cross validation;

      while *number of dataset splits* < 5 **do**

        split the dataset into *trv* with balanced main-class and sub-domain distribution and *rem*;

        compute recall by main-class in total and by sub-domain;

        compute chi-squared p-value from confusion matrices for sub-domains;

        train classifier on full *trv* set and classify the full *rem* set;

        while *number te samples* < 5 **do**

          for *te main-class proportion in* [0.1 *to* 0.9] **do**

            for *te sub-domain proportion in* [0.1 *to* 0.9] **do**

              sample a *te* set from *rem* with replacement;

              compute class proportions in *te* using sd and nsd-methods;

              write results to file;

            end

          end

        end

      end

    end

  end

end

Algorithm 4: Exploration of sd and nsd quantification methods on UCI Datasets

This generated a large set of results that were then randomly sampled to give 400,000 results per dataset.

4.2.4 Parameters

Table 4.5: Parameter settings

Parameter	Value	Notes
Training set size	10^2 to 10^4	See Fig. 4.4
Training set class 0 proportion	0.5	
Test set size	10^3	
Test set class 0 proportion	0.1 to 0.9	Steps of 0.1
Classifier type	SVM	
Classifier kernel	Linear	
C values	$\{10^{-4}$ to $10^1\}$	Chosen by CV[6]

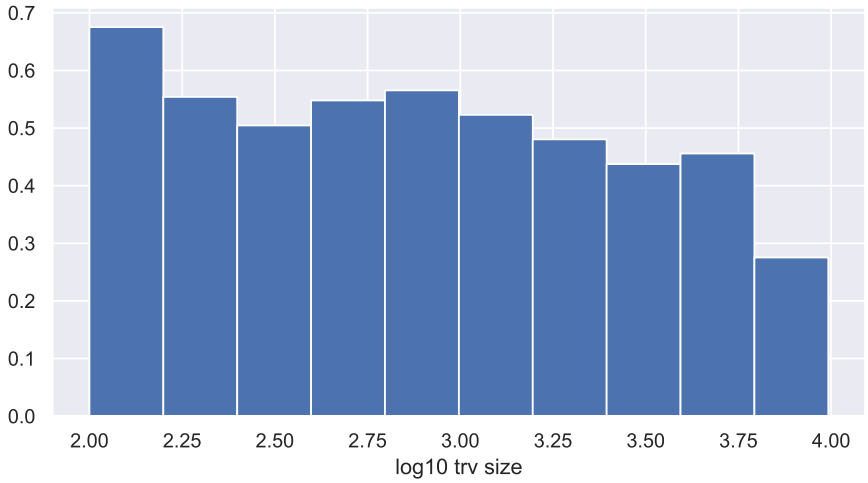

Figure 4.4: Distribution of trv set sizes

[6]The C value was chosen based on accuracy with 5-fold cross validation on the training-validation set

4.2.5 Classifier performance

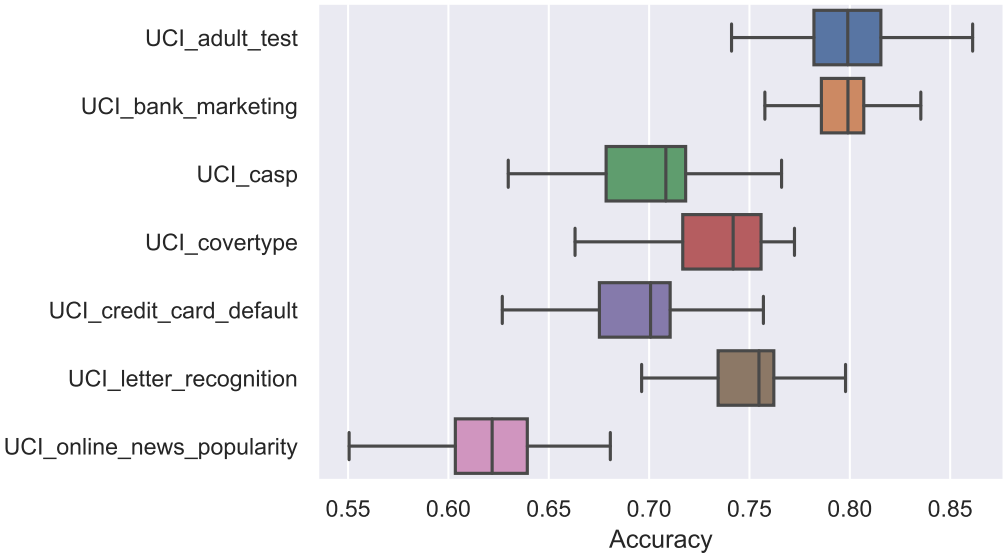

Figure 4.5: Classifier accuracy by dataset

With reference to Figure 3.10: the accuracy of the classifier on 6 of the 7 UCI datasets should be sufficiently high to ensure acceptable quantification accuracy. The lower accuracy of the classifier for the Online New Popularity dataset could lead to low quantification accuracy, particularly with training sets of close to 100 instances. However, the focus of this section is on the differential performance of two quantification methods and any any poor performance with Online New Popularity is not apparent in the results in Section 4.3.3.

4.2.6 Results

4.2.6.1 Size of validation dataset

Figure 4.6 shows the impact of the size of the training-validation set (trv) on the absolute performance of the sd and nsd-methods.

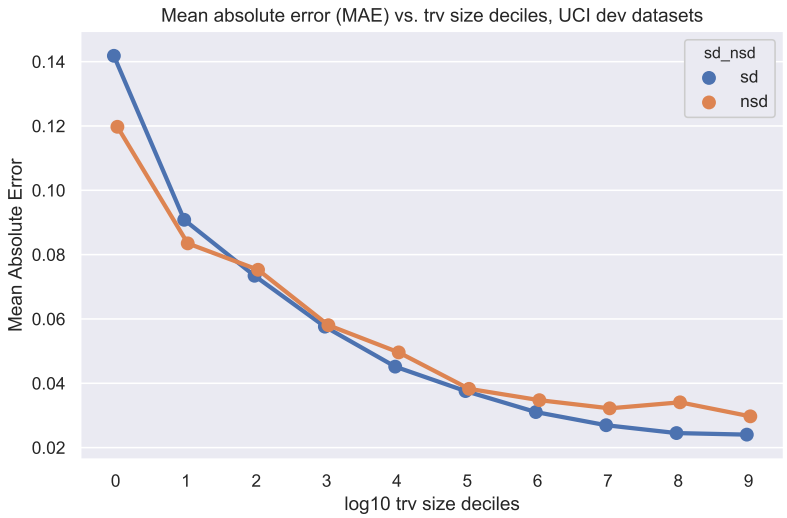

Figure 4.6: Mean delta absolute error nsd-method minus absolute error sd-method by decile of trv size. UCI dev datasets. 95% confidence intervals

Figure 4.7 plots the same data but shows the *differential* performance of the sd and nsd-methods for each of the UCI_dev datasets separately.

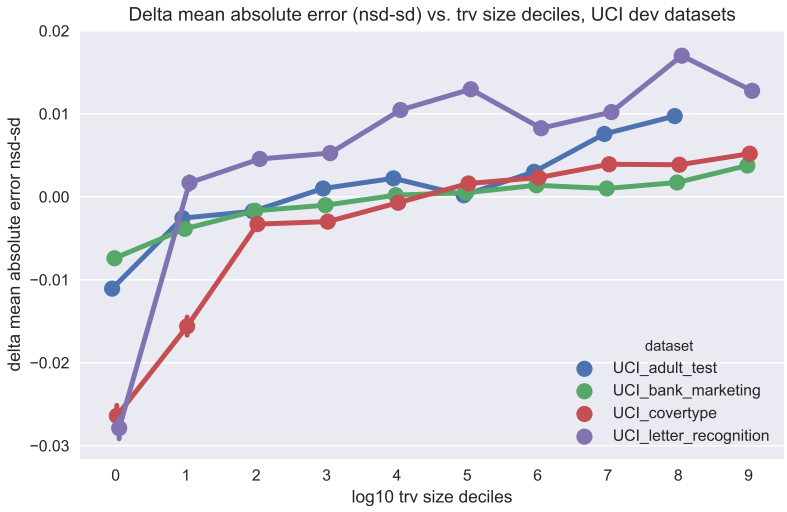

Figure 4.7: Mean delta absolute error nsd-method minus absolute error sd-method by decile of trv size. UCI dev datasets. 95% confidence intervals

The four UCI_dev datasets show the same relationship as was seen with the simulated data in Sections 3.8.3 and 4.1.3.1, i.e. when the validation dataset is small the nsd-method gives better mean quantification accuracy than the sd-method but once the validation dataset is above a certain size, on average the sd-method out-performs the nsd-method. The initial

hypothesis was that it would be difficult to find a value for the size of the validation set that was 'sufficiently large' for all datasets. However, figure 4.7 shows that for all four UCI_dev datasets, when the validation set was in the 5th decile or above[7] (above ≈ 1000 instances) then, on average, the sd method was better than the nsd method.

4.2.6.2 Probability of difference in recall by sub-domain

Figure 4.8 shows the relationship between the significance of the difference in recall at sub-domain level (expressed through log10_c2p_sum) and the absolute performance of the sd and nsd-methods on the UCI_dev datasets overall.

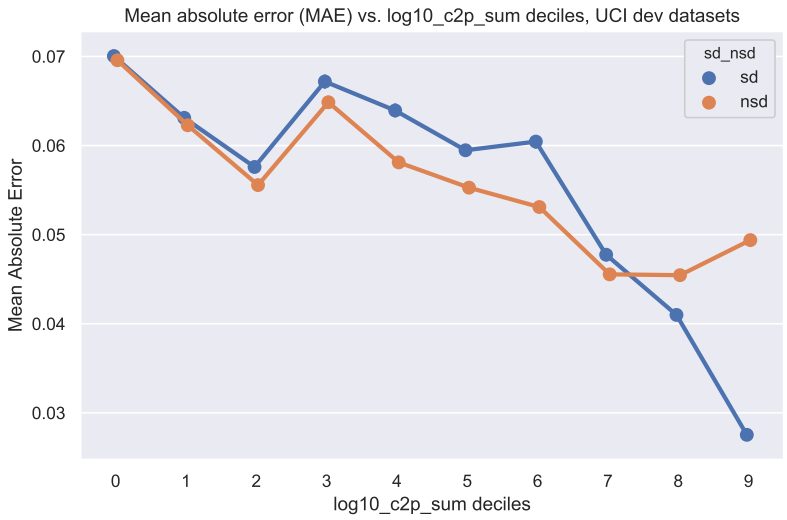

Figure 4.8: Mean delta absolute error nsd-method minus absolute error sd-method by decile of log10_c2p_sum. UCI dev datasets. 95% confidence intervals

Figure 4.9 again plots the same data but shows the relationship between the significance of the difference in recall at sub-domain level (expressed through log10_c2p_sum) and the relative performance of the sd and nsd-methods on each UCI_dev dataset separately.

[7]Deciles 4 and 5 in Figure 4.7 span a trv_size range from 489 to 1,148

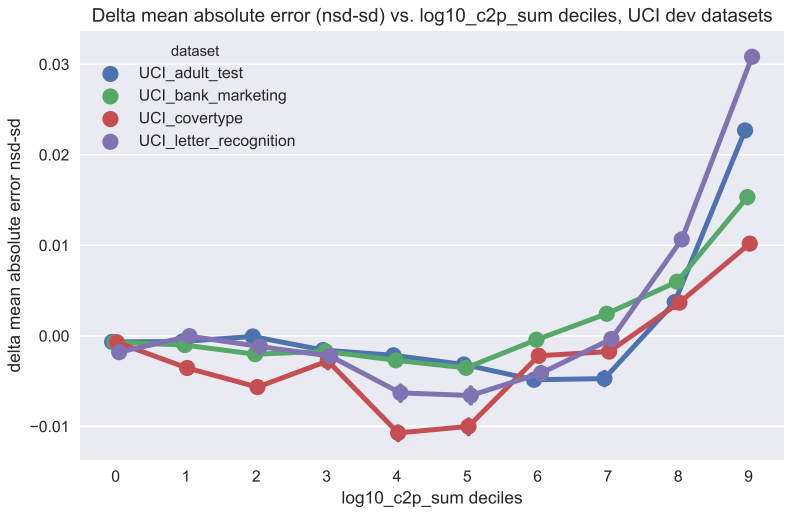

Figure 4.9: Mean delta absolute error nsd-method minus absolute error sd-method by decile of log10_c2p_sum. UCI dev datasets. 95% confidence intervals

Again, this is consistent with the observations made with simulated data in Section 4.1.3.3. When the statistical significance of the recall difference is high (high values of log10_c2p_sum) the sd-method outperforms the nsd-method. However, for all four UCI_dev datasets this improvement only occurs in the top 2 deciles i.e. for only 20% of the results.

4.2.6.3 Difference in sub-domain proportion

Figure 4.10 shows the relationship between the difference in sub-domain proportions between the trv dataset and the te dataset and the relative performance of the sd and nsd-methods.

The concept of *sub-domain distance (SDD)* was introduced in Section 4.1.3.3. In this experiment the sub-domain label of the data instances in the test set is known but, as we are looking for a method that can be used in practice we are assuming that we do *not* know their main-class label and therefore we cannot use *SDD*. Instead we simply measured the absolute difference between sub-domain proportions in the trv dataset (always set to 0.5 in this experiment) and the sub-domain proportions in the test set (ranging between 0.1 and 0.9) regardless of main-class. This metric was termed *SDDN*.

Figure 4.10 shows the relationship between mean absolute error and *SDDN*.

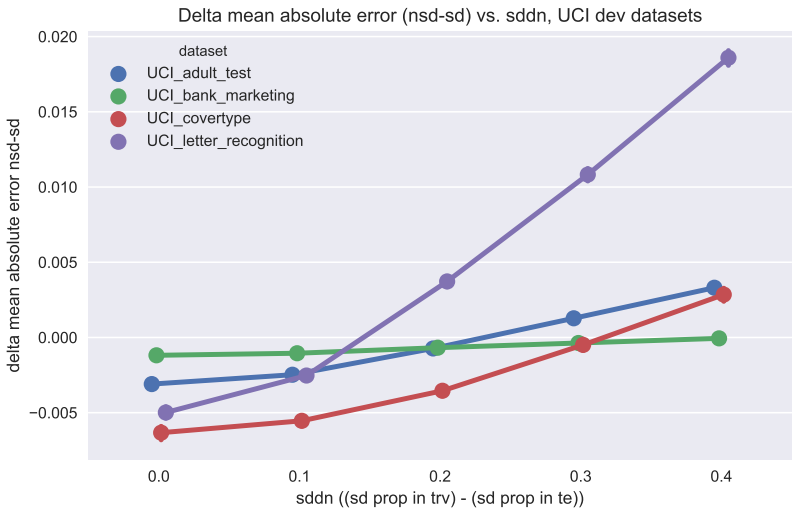

Figure 4.10: Mean delta absolute error nsd-method minus absolute error sd-method by absolute difference in sub-domain proportion between trv and te (SDDN). UCI dev datasets. 95% confidence intervals

Again, this is consistent with the observations made with simulated data in Section 4.1.3.3, when *SDDN* is small the nsd-method outperforms the sd-method. However, for most of the datasets the effect is small even at high values of SDDN. When SDDN is small, in all cases there was, on average, no advantage to using the sd-method.

4.2.7 Discussion

The results with the UCI_dev datasets are consistent with the simulation findings in Section 4.1. The statistical significance of the recall difference by sub-domain (log10_c2p_sum) and the difference in sub-domain proportions between the trv set and the test set (SDDN) *can* be used as thresholds for the application of sd-method *but* they only identify a small proportion of the total cases where applying the sd-method would be advantageous.

As a single parameter, the size of the trv set itself appears to be a better measure for identifying when using the sd-method will, on average, be advantageous. The hypothesis at the start of this section was that it would potentially be difficult to identify a value for trv size that would be 'sufficiently large'. However, as shown in Figure 4.7 and previously with simulated data in Figure 4.1, when the validation set size is above around 1000 the sd-method on average gives a higher quantification accuracy than the nsd-method.

4.3 Experiment 3: determining thresholds

Contrary to initial expectations, the conclusion from Section 4.2 was that the size of the trv set was the best *single* measure against which to set a threshold for the application of the sd-method. However, while they are perhaps not as good individually, *log10_c2p_sum* and *SDDN* could potentially be included in a multi-criteria threshold. *log10_c2p_sum* appeared to be a good measure for identifying a small number of cases where large benefits might be obtained while *SDDN* appeared to be a good measure for identifying cases where the sd-method should *not* be applied. In this section, the aim is to see if a multi-criteria threshold can give a better performance than just using the size of the trv set alone.

As was seen earlier, it is possible to set the criteria-thresholds to get a large improvement on a small proportion of results or to set them to get a smaller improvement over a larger proportion of results. The chosen method for measuring quantification accuracy was to look at the mean improvement over *all* results e.g. a 2% improvement that applied to 50% of the results would be a 1% mean improvement over all results. In addition, the criteria-thresholds should ideally be set to minimise the likelihood of using the sd-method when it may *reduce* performance so *proportionate performance* is also taken into consideration.

In the tables below ΔMAE is the shortened term for 'delta mean absolute error nsd-sd'.

4.3.1 Single criteria

The results with trv_size as the single criteria are given in Table 4.6:

Table 4.6: Difference in abs error between nsd and sd-methods for various values of baseline $\log_{10}$ trv_size. UCI_dev datasets

$\log_{10}$ trv_size	2.0	**2.5**	3.0	3.5
Number of results above threshold	1,597,502	1,117,226	684,635	267,488
Proportion of results above threshold	100.0%	69.9%	42.9%	16.7%
ΔMAE absolute, results above threshold	0.0009	0.0047	0.0065	0.0083
ΔMAE absolute, all results	0.0009	**0.0033**	0.0028	0.0014

A $\log_{10}$ trv_size of $10^{2.5}$ (i.e. 316) gave the highest ΔMAE mean average over all results.

4.3.2 Multiple criteria

Using a grid search the optimum parameter values were found to be those shown in Table 4.7:

Table 4.7: Optimum multiple parameter values. UCI_dev datasets

	Parameter	Value
	trv_size	> 316
AND	log10_c2p_sum	> 1.71
AND	SDDN	≥ 0.1

Table 4.8 shows the performance using the single and the multiple criteria on the UCI_dev datasets:

Table 4.8: Comparison of quantification performance of single and multiple criteria on the UCI_dev datasets

UCI_dev			
Number of results	959,498		
Criteria	Baseline	Single	Multiple
trv_size		>316	>316
log10_c2p_sum		all	>1.71
SDDN		all	≥0.1
All results			
MAE mean	0.0560	0.0529	0.0528
ΔMAE mean, absolute improvement		**0.0031**	**0.0032**
ΔMAE mean, relative improvement		5.4%	5.6%
Results above threshold			
Number		654,646	299,236
Proportion		68%	31%
ΔMAE mean, absolute improvement		0.0030	0.0101

On the UCI_dev collection of 4 datasets the multiple criteria are only marginally better than the method using just the validation dataset size. However the multiple criteria method is more selective, selecting less than half the number of results for use with the

sd-method. This is explored further in Section 4.3.3.1.

4.3.3 Evaluation against the held-out UCI_test datasets

Table 4.9 shows the results for the combined UCI_test datasets on the optimal criteria that were set on the UCI_dev datasets:

Table 4.9: Combined UCI_test datasets. Quantification performance with single and multiple criteria.

All UCI_test Datasets			
Number of results	720,000		
	Baseline	Single	Multiple
All results			
MAE mean	0.1128	0.1086	0.1077
ΔMAE mean, absolute improvement		**0.0042**	**0.0051**
ΔMAE mean, relative improvement		3.7%	4.5%
Results above threshold			
Number		513,361	244,325
Proportion		71%	34%
ΔMAE mean, absolute improvement		0.0059	0.0150

Tables 4.10, 4.11 and 4.12 show the same results as Table 4.9 broken down by individual UCI_test datasets:

Table 4.10: UCI_CASP dataset. Quantification performance with single and multiple criteria.

CASP			
Number of results	240,000		
	Baseline	Single	Multiple
All results			
MAE mean	0.0784	0.0728	0.0728
ΔMAE mean, absolute improvement		**0.0056**	**0.0056**
ΔMAE mean, relative improvement		7.1%	7.1%
Results above threshold			
Number		180,163	120,371
Proportion		75%	50%
ΔMAE mean, absolute improvement		0.0075	0.0112

Table 4.11: UCI_credit_card_default dataset. Quantification performance with single and multiple criteria.

Credit Card Default			
Number of results	240,000		
	Baseline	Single	Multiple
All results			
MAE mean	0.0960	0.0940	0.0933
ΔMAE mean, absolute improvement		**0.0020**	**0.0027**
ΔMAE mean, relative improvement		2.1%	2.8%
Results above threshold			
Number		165,236	63,730
Proportion		69%	27%
ΔMAE mean, absolute improvement		0.0030	0.0103

Table 4.12: UCI_online_news_popularity dataset. Quantification performance with single and multiple criteria.

Online News Popularity			
Number of results	240,000		
	Baseline	Single	Multiple
All results			
MAE mean	0.1633	0.1584	0.1565
ΔMAE mean, absolute improvement		**0.0048**	**0.0068**
ΔMAE mean, relative improvement		2.9%	4.2%
Results above threshold			
Number		168,226	60,465
Proportion		70%	25%
ΔMAE mean, absolute improvement		0.0068	0.0270

Using multiple criteria for selecting when to use the sd-method gives better overall result than just using the single criteria of trv_size on all three UCI_test datasets. Using a single criteria of trv_size itself also outperformed the nsd-method on all three UCI_test datasets.

4.3.3.1 Proportionate performance

Ideally the selection criteria would be perfectly discriminative and would select *all results* where the sd-method is going to be better than the nsd-method and *no results* where the nsd-method is going to be better.

Table 4.13 shows the proportion of cases where the sd-method was chosen (i.e. the criteria-threshold was met) *and* where the sd-method gave a lower quantification error than the nsd-method.

Table 4.13: Proportion of results where the nsd-method gives larger error than the sd-method

Dataset	Proportion nsd>sd Full Dataset	Proportion nsd>sd Selected by Single Criteria	Proportion nsd>sd Selected by Multiple Criteria
CASP	51%	55%	58%
Credit Card Default	46%	50%	57%
Online News Popularity	45%	50%	62%
Overall	48%	52%	59%

This shows that while the proportion of results where the sd-method is better than the nsd-method is higher using the single criteria of validation set size than in all results as a whole, it is higher again using multiple criteria. Multiple criteria are more discriminative than the single criteria.

4.3.4 Discussion

In this experiment, using a threshold for applying the sd-method based *only* on trv_size did result in better mean quantification accuracy on all three of the UCI_test datasets when compared with the nsd-method. However, using a threshold criteria made up from a combination of trv_size, statistical likelihood of a recall difference by sub-domain (*log10_c2p_sum*) and difference in the proportion of sub-domains (*SDDN*) gave a better mean quantification performance than when the criteria was only on trv_size. Using multiple criteria gave a mean absolute improvement in MAE of 1.54 percentage points on the 34% of cases that met the criteria equating to a mean absolute improvement of 0.51 percentage points overall with a range of 0.27 to 0.68 percentage points over the three UCI_test datasets.

4.4 Experiment 4: Twitter dataset

The aim in this section is to revisit Twitter users and see if the tsd-method can deliver similar improvements in quantification accuracy to those seen with UCI datasets in Section 4.3.

4.4.1 Twitter Age Friends (TAF) dataset

A new, and significantly larger Twitter dataset was built for this experiment. Twitter provide a free stream of a 1% random sample of all tweets. This was sampled between February 2017 and April 2018 using the Method52[8] tool. The users that generated the tweets were added to the dataset. Each user would appear just once in the dataset regardless of how many of their tweets were picked up so sampling over such a long period of time should reduce the potential bias in the dataset towards users that send tweets the most frequently.

The method of building the dataset was broadly in line with that described earlier in Section 3.4. The Tweet and the attached user details were selected if the user screenname ended in 4 digits that were between 1940 and 1999 inclusive[9]. The IDs of the Twitter accounts that were being followed ('friends') by these users were collected using API[10] calls in Method52. This generated in a file of 274,000 users and a file of 126M user-friend pairings. Method52 was then again used to label ('annotate') users by gender, social-class, location and presence of children (POC). These annotation labels were then used as our sub-domain labels.

The last four digits of the user screenname were taken as a plausible year of birth (see Section 3.4.1.1). To make this into a classification problem the dataset was split on the median year of birth of 1984. Users from years 1983, 1984 and 1985 were removed from the dataset to give a separation between the classes. Age label 0 was assigned to users with an estimated year of birth before 1983 and an age label of 1 was assigned to users with an estimated year of birth after 1985. The distribution by estimated year of birth in the finished TAF dataset is shown in Figure 4.11.

[8] Courtesy of CASM Consulting LLP

[9] Potentially indicative of year of birth

[10] Application Programming Interface

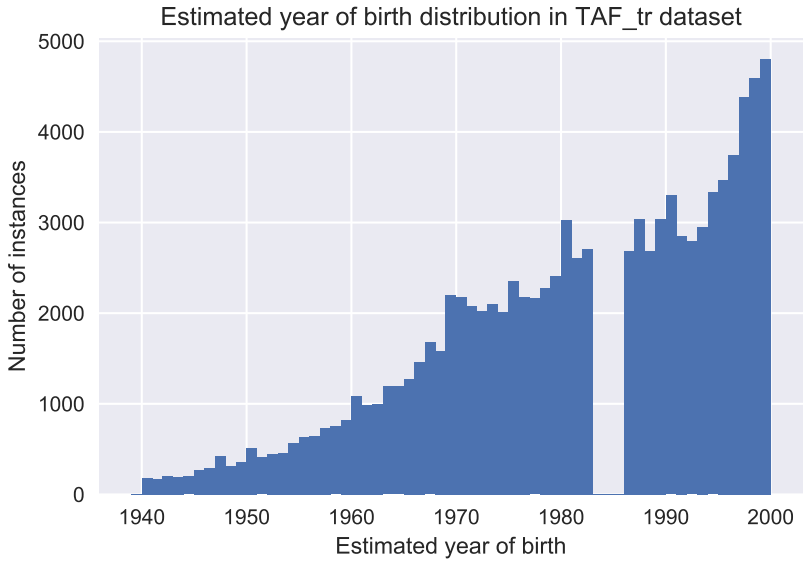

Figure 4.11: Distribution by estimated year of birth in the TAF training dataset

Table 4.14 lists the further filtering was applied to 'clean' the dataset.

Table 4.14: Filtering applied in the generation of the TAF dataset

	Filter	Rationale
1	>2000 friends	Likely not to be individuals
2	>2000 friends	Likely not to be individuals
3	Classified as 'institution' by Method52	Likely not to be individuals
4	Tweet and user language is not English[11]	Accounts followed are likely to be language specific

In Twitter terminology an account that is being followed is called a 'friend'. Following an account results in the user seeing the Tweets that are made from that account. Any Twitter user can follow any other Twitter account. Using accounts followed (friends) as features resulted in very high dimensionality (initially 14M features in this case). Dimensionality reduction was applied to reduce the number of features to a manageable level while maintaining classification accuracy. The dimensionality reduction steps are given in Table 4.15. Accuracy was measured by training a linear kernel SVM classifier on 10,000

[11] Language codes: en, en-AU, en-GB, en-IN, en-US, en-gb, und

instances selected at random from TAF_tr and then tested on 1,000 instances selected at random from TAF_dev.

Table 4.15: Dimensionality reduction steps in the generation of the TAF dataset

	Processing	Number of Features	Accuracy
1	Remove friends followed by < 50 users in the dataset	108,421	0.744
2	Chi-squared feature selection	20,000	0.735
3	Principal Component Analysis	200	0.748

This gave a dataset of 174,667 user instances which was split into three datasets by random sampling:

Table 4.16: TAF dataset split into tr, dev and te

	Purpose	Size
TAF_tr	Training	100,000
TAF_dev	Test and validation while developing	54,667
TAF_te	Held-out for final testing	20,000

4.4.2 Dataset initial analysis

Main-class recall can vary by sub-domain, as shown in Chapter 3. Method 52 was used to annotate the Twitter users in the TAF dataset by gender, social class, location (continent) and presence of children in the household (POC). A linear-kernel SVM classifier was trained on a random sample of 5,000 instances from the TAF_tr dataset. This was then used to classify all the data in the TAF_dev dataset. Recall by main-class for the classifier was computed overall ('All') for the full TAF_dev dataset then separately for each sub-domain.

Figure 4.12 shows the differences in recall for classes 0 and 1 across those four attributes against the baseline recall figures for all the instances in the TAF_dev dataset.

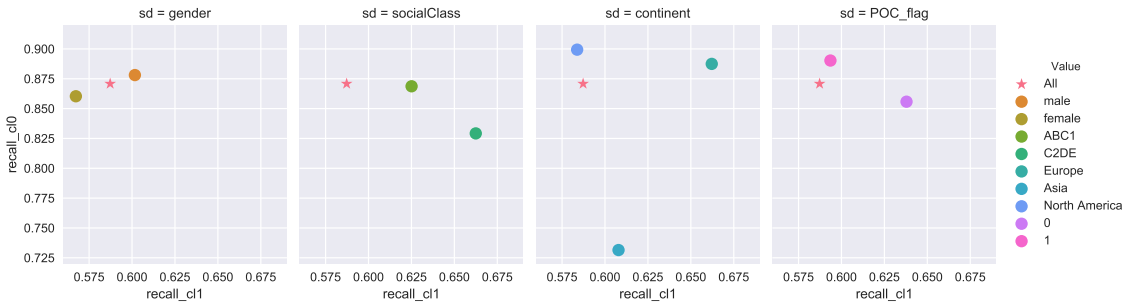

Figure 4.12: Mainclass recall values by subset of the TAF_dev dataset

The largest differences in recall appear between users estimated to be in different continents. For class 0, recall for users located in Asia is 0.731 compared to 0.899 for users located in North America and 0.871 for the TAF_dev dataset as a whole.

These differences in recall values illustrate the problem at the core of this thesis: how to quantify accurately when the distribution of the dataset being quantified does not match that on which the quantifier was trained. For example, a *classify and adjust* quantifier that is trained on the TAF dataset and which (directly or indirectly) uses the performance figures for the classifier obtained on the whole TAF dataset would potentially give an inaccurate estimate of class proportions if it was applied to a set of users located in Asia.

4.4.3 Method

The method was the same as for the previous experiments with the UCI_dev datasets with the exception that the annotated values for gender, social class, location (continent) and presence of children in the household (POC) were used for indicating sub-domain instead of using binary features from the dataset itself.

The program was run 8 times generating a results dataset of 1,004,400 separate sample experiments.

4.4.4 Results

Table 4.19 shows the results from applying the tsd-method with the criteria and thresholds set previously on the UCI_dev datasets (see Table 4.8) to the TAF dataset:

Table 4.17: UCI_online_news_popularity dataset. Quantification performance with single and multiple criteria.

TAF			
Number of results	240,000		
	Baseline	Single	Multiple
All results			
MAE mean	0.0981	0.0978	0.0975
ΔMAE mean, absolute improvement		**0.0003**	**0.0007**
ΔMAE mean, relative improvement		0.3%	0.7%
Results above threshold			
Number		178,302	71,701
Proportion		74%	30%
ΔMAE mean, absolute improvement		0.0004	0.0022

Similarly - looking at the proportion of results within the selection criteria where the sd-method was better than the nsd-method:

Table 4.18: Proportion of results where the nsd-method gives larger error than the sd-method, TAF dataset

TAF	Proportion nsd>sd	Proportion nsd>sd Selected by	Proportion nsd>sd Selected by
	Full Dataset	Single Criteria	Multiple Criteria
Overall	45%	48%	53%

Both the single selection criteria of validation set size and the multiple criteria give mean improvements in quantification performance with the TAF dataset but the improvements are *considerably smaller* than were seen with the UCI_test datasets in Section 4.3.3. This could be because the thresholds set on the UCI_dev dataset do not work with the TAF dataset *or* because the scope for improvement in quantification accuracy improvement with the sd-method on the TAF dataset is limited. To test this thresholds were set on the TAF dataset itself (TAF optimum). The comparison between thresholds set using UCI_dev dataset and thresholds set using the TAF dataset itself are shown in Table 4.19.

Table 4.19: TAF dataset. Quantification performance with single and multiple criteria.

TAF					
	Baseline	Single	Single	Multiple	Multiple
Criteria optimised on:		UCI_dev	TAF	UCI_dev	TAF
trv_size		>316	>1,000	>316	>1,000
log10_c2p_sum		all	all	>1.71	>0.22
SDDN		all	all	≥0.1	≥0.2
All results					
MAE mean	0.0981	0.0978	0.0971	0.0975	0.0970
ΔMAE mean, absolute		**0.0003**	**0.0010**	**0.0007**	**0.0011**
ΔMAE mean, relative		0.3%	1.0%	0.7%	1.1%
Results above threshold					
Proportion		74%	43%	30%	28%
ΔMAE mean, absolute		0.0004	0.0023	0.0022	0.0038

Comparing quantification performance on the TAF dataset with selection criteria optimised on both UCI_dev and on TAF itself shows that using multiple criteria gets closer to its TAF-optimal performance than using the single criteria of validation set size.

4.4.5 Discussion

The results on the TAF dataset were an order of magnitude *worse* than those obtained on the UCI_test datasets. When the multiple criteria based method identified that explicit sub-domains should be used this gave better quantification accuracy in only 53% of cases with the TAF dataset as compared to 59% of cases with the UCI_test datasets. This may be due to the method of biassing. The TAF dataset was biased using the annotated sub-domains shown in Figure 4.12. Some of those categories have large difference in recall, e.g. 'Asia' in location, but many do not. It may simply be that the test sets were not particularly biased. This hypothesis is supported by the finding that the set thresholds achieved a performance that was 55% of the *optimal* performance for the dataset i.e. there was not actually much room for improvement above the baseline which would be consistent with a low level of bias in the test sets.

4.5 Conclusions

A summary of the results of the various experiments are given in Table 4.20.

Table 4.20: Summary of Chapter 4 results

		Method	Set On	Measured On	ΔMAE Absolute	ΔMAE Relative
Tab.	4.9	tsd single	UCI_dev	UCI_test	0.42%	3.7%
Tab.	4.9	tsd multiple	UCI_dev	UCI_test	0.51%	4.5%
Tab.	4.19	tsd single	UCI_dev	TAF	0.03%	0.3%
Tab.	4.19	tsd multiple	UCI_dev	TAF	0.07%	0.7%

In Chapter 3 it was shown that a quantification method that uses explicit sub-domains (the sd-method) *can* give better quantification accuracy than one that does not (the nsd-method). In this chapter it has been possible to use threshold-criteria based on parameter values that would be known a priori to determine when explicit sub-domains should and should not be used. These give an improvement in quantification accuracy on held-out test data in all cases. Using multiple criteria applied to the UCI_test datasets produced relative improvements in mean absolute error in quantification accuracy of between 2.8% and 7.1%. However, the improvement over the baseline nsd-method with the TAF dataset of Twitter users was much more modest. This may be due to lower levels of bias in the test sets than was the case with the UCI_test datasets.

A key limitation of the explicit sub-domains approach is that the sub-domains have to be identified in advance. No approach is put forward for finding optimal sub-domains. In reality, sub-domains would most likely be chosen in a trial and error process using the data labels that were available. Furthermore, the biassing of the test datasets was made using the *same* explicit sub-domains. As a result, the improvements in quantification accuracy that were found are likely to be towards the upper bound.

In the following chapters, methods are put forward do not require any up-front knowledge of the source of potential dataset shift and are these are tested on test datasets with a broad range of biasing.

Chapter 5

Domain adaptation by instance weighting

In Chapters 3 and 4, an improvement in quantification accuracy under class-conditional dataset shift was achieved by using explicit sub-domains for domain adaptation. A relative improvement in Mean Absolute Error (MAE) of 4.5% was obtained on the UCI_test datasets. However, these methods require that the sub-domains on which the dataset shift has occurred are known in advance and are explicitly labelled in the training and test data.

The approach in this chapter is more general. No prior knowledge of the cause, direction or scale of the dataset shift is required. Correction for class-conditional dataset shift is made solely on the basis of the observed difference in the distribution of the data between the set that is being tested and the set used for training.

The methods in this chapter are based on the *instance weighting* methods for domain adaptation. Weights are computed for each instance in the training set using the data from the training and test sets. The instances in the training set that are assigned higher weights can be considered to be 'close' to the instances in the test set. The weighted distribution of the training set (drawn from the Source domain) should be the same, or very similar to, the unweighted distribution of the test set (drawn from the Target domain). In this chapter the aim is to see if these instance weighting methods can be adapted so that they reduce or eliminate *class-conditional* dataset shift. If they do, then the *classify and adjust* quantification methods that rely on the *class-conditional* feature distributions in the Target domain being similar to those in the Source domain should be effective at estimating class proportions in the Target domain.

The two main importance weighting methods used in this chapter are Kernel Mean Matching (KMM) and Unconstrained Least Squares Importance Fitting (uLSIF). Overviews of these methods are given in Sections 5.4 and 5.5 respectively. Section 5.6 explores the relationship between class-proportions in the test set and the distribution of instance weights. In Section 5.7 the KMM and uLSIF methods are used with a matrix-inversion *classify and adjust* quantifier. An alternative *instance weighting* method, Sample Selection Bias Correction (SSBC), is explored in Section 5.8, while an iterative derivative of this is explored in Section 5.9. Finally the chapter conclusions are in Section 5.10.

5.1 Measuring dataset shift

Quantification is about estimating the class proportions in a test set. The assumption is that these class proportions can vary without constraint, that one class can make up anywhere between 0% and 100% of the test set. Standard measures of dataset shift (see Section 2.2.3) simply compare the *overall* distributions. A difference due simply to the class distribution that we are trying to estimate, with no difference in class-conditional feature distribution, would still be seen as *dataset shift* under these measures.

Consider a toy example. There are two classes 0 and 1. Set A consists of 90% instances of class 0, 10% class 1. Set B consists of 10% instances of class 0 and 90% class 1. A classifier can distinguish between class 0 and class 1 with high accuracy. All the class 0 instances in both A and B have been drawn iid from all the class 0 instances in the same domain. The same with class 1. Class-conditionally, there is no difference between A and B *but* the distribution of A *is* quite different from B. As there is no class-conditional difference a standard *classify and adjust* quantification method should work well despite the presence of *dataset shift*.

Ideally we would like a measure of *class-conditional* dataset shift. To do this, we compare training and test sets that have the same class proportions. With no difference in class proportions, the difference between the two sets should then be due to differences in class-conditional feature distribution. Sub-samples are taken from the training data of the same class-proportions as the test set and Proxy $\mathcal{A}$-distance (PAD) (see Section 2.2.3.1) is measured between the two sets. This is repeated a number of times with different random samples of the training set and the values of Proxy $\mathcal{A}$-distance are averaged. I have termed the resulting value *class-balanced* proxy $\mathcal{A}$-distance or PADcb for short. Clearly, as this

requires knowledge of the class distribution in the test set, this cannot be used as part of a *method* but it can be used in *analysis*.

5.2 Datasets

The UCI and TAF datasets from Chapter 4 were used. The method used for generating the biassed[1] test sets was taken from Gretton et al. [58] (which itself was originally from Zadrozny [122]). Test set instances were selected from the dataset based on comparing the value of a randomly chosen feature against a value drawn from a normal distribution of random mean and random variance. The test sets were built to a given class proportion which was selected randomly in the range [0.04, 0.96]. This range was chosen to give both a wide range of class proportions but also to minimise the impact of clipping class proportions into [0,1]. As well as original features Gretton et al. [58] also used the first PCA component for biassing. I was concerned that the first PCA component might have a strong correlation with class so I used the first *five* PCA components.

Figure 5.1 shows the distribution of the test sets by *dataset shift* as measured by PADcb.

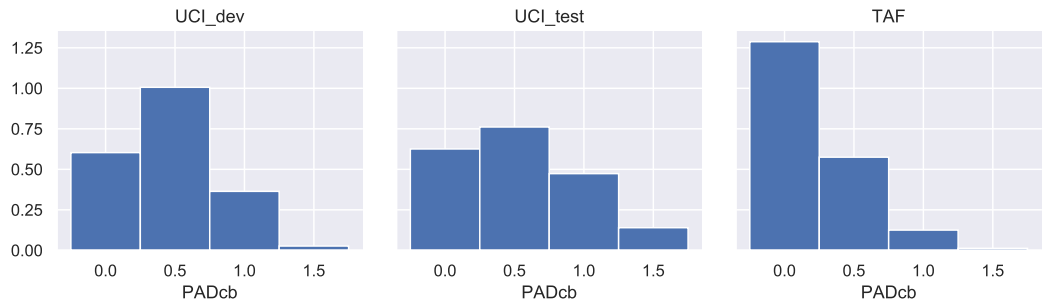

Figure 5.1: Typical normalised distribution of the level of dataset shift in the test sets drawn from UCI_dev, UCI_test and TAF. Dataset shift measured by PADcb.

Despite the biassing method and parameters being identical for all datasets, the outcome in terms of distribution of test sets by dataset shift is clearly quite different.

The distribution of test sets by dataset shift for the individual datasets within UCI_dev is

[1] i.e. with dataset shift relative to the original dataset from where the instances were sampled

shown in Figure 5.2 and within UCI_test in Figure 5.3.

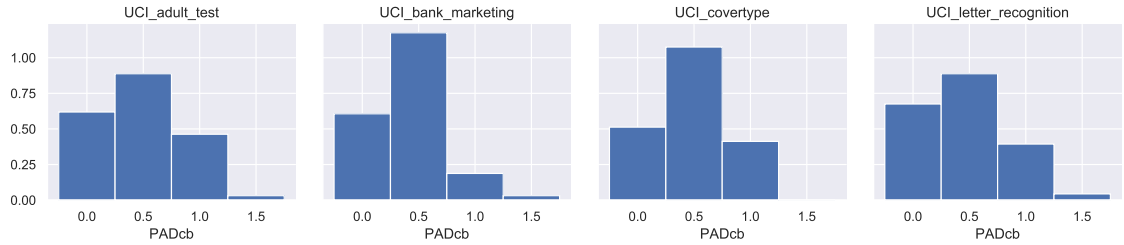

Figure 5.2: Typical distribution of the level of dataset shift in the UCI_dev datasets test sets. Dataset shift measured by PADcb.

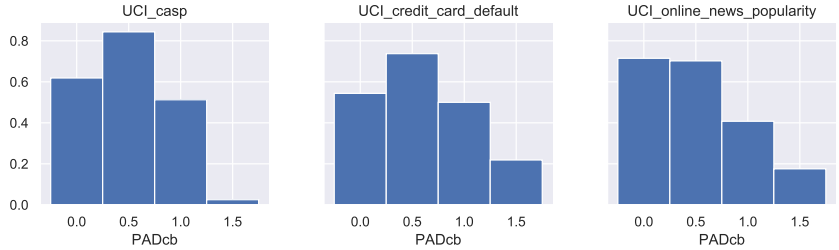

Figure 5.3: Typical distribution of level of dataset shift in the UCI_test datasets test sets. Dataset shift measured by PADcb.

Again, while all the datasets were biased using a common method and common parameters, there were obvious differences in the distribution of dataset shift in the different datasets.

5.3 Instance weighting

The instance weighting approaches used in this chapter rely on the assumption that the dataset shift that is being dealt with can be characterised as *covariate shift*. If the Target domain from which the training set is drawn is within the span of the Source domain from which the test set is drawn, and the feature space X contains information from which the identity of the Target domain could be reasonably estimated then this assumption appears reasonable.

With the covariate shift assumption $P_T(y|x) = P_S(y|x)$, the weights w_i are the ratio of the densities in the two domains as previously given in Equation 2.8:

$$w_i = \frac{P_T(x_{Si})}{P_S(x_{Si})}. \tag{5.1}$$

The various means of computing the weights w_i are discussed in Section 2.3.2.

5.4 Kernel Mean Matching (KMM)

Kernel Mean Matching is a well-known and widely-cited method for direct density ratio estimation that was originally put forward in Huang et al. [66] and Gretton et al. [58]. The principle behind the method is to project the training and test data into a Reproducing Kernel Hilbert Space (RKHS) and compute the weights for the training data that minimise the difference in the means of the training and test sets in the RKHS.

The original authors provided Matlab code for the KMM method on their website. I converted this to Python and verified the Python conversion against their Matlab original with synthetic data. CVXOPT [7] was used for solving the convex optimisation problem. Further details of the KMM method are given in Appendix B.2.

The KMM method has two parameters, the kernel size (*sigma*) and the number of kernels (B). A rule-of-thumb that is sometimes applied (e.g. [58]) with Gaussian kernels is to set the kernel size equal to the median L_2-norm distance between instances. Rather than simply rely on this rule-of-thumb, in this experiment the median L_2-norm distance between instances in is used as the *basis* kernel size for each dataset. The kernel size was then set as a *multiple* of this basis size with multiples of 0.01, 0.1, 1 or 10. The number of kernels (B) was either 160, 500 or 1600. The training set size was 1600 instances.

Figure 5.4 shows how the distribution of weights varies with *sigma* and B for the four UCI_dev datasets. The y-axis gives *median cumulative weight proportion*, i.e. the proportion of the total sum of weights that is assigned to instances whose weight is above the median weight. A figure of 0.5 implies that all the instances are equally weighted. A figure approaching 1.0 implies that a small number of instances have a very high weighting.

Figure 5.4 shows that a kernel size of 0.01 times the median L_2-norm distance between instances gives an unreliable allocation of weights with large differences in weight allocation between datasets. A kernel size multiplier in the range of 0.1 to 10 appears to give a more consistent distribution of weight to the instances with more concentration of weight on a smaller number of instances at higher levels of dataset shift. The variation in that distribution (as can be observed by the slope of the correlation line) is greater for higher kernel sizes (multiple of 10) than for small kernels (multiple of 0.1).

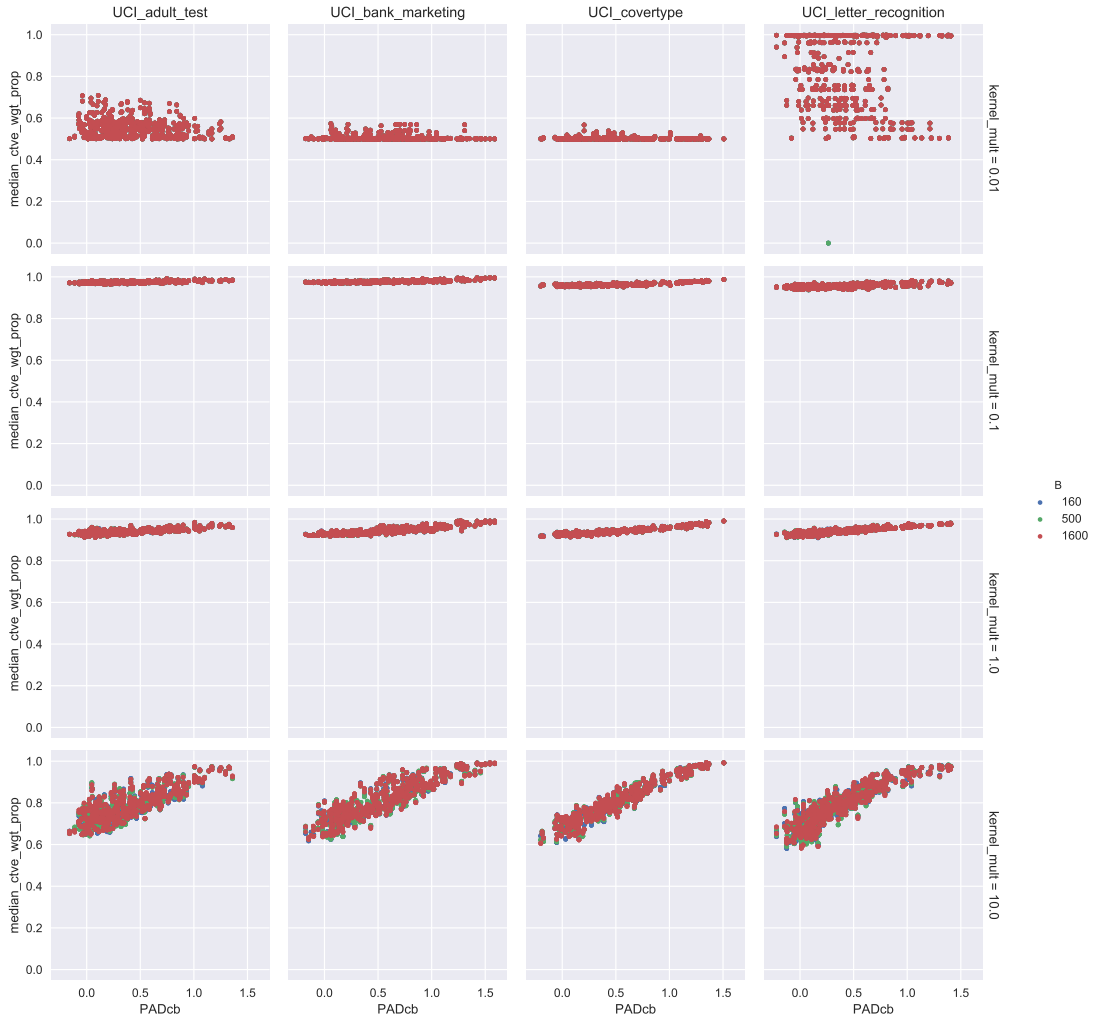

Figure 5.4: Median cumulative weight proportion from KMM method vs. level of dataset shift as measured by PADcb. Shown separately by dev dataset, kernel size multiple and B parameter

5.5 Unconstrained Least Squares Importance Fitting (uLSIF)

Unconstrained Least Squares Importance Fitting was developed more recently than Kernel Mean Matching. Sugiyama and Kawanabe [104] claim that it gives superior performance. With uLSIF, instance weights are computed in closed-form so it is considerably quicker than methods that require the optimisation of a function such as KMM.

As with KMM, the original authors provided Matlab code for the method on their website. Again, I converted this to Python and verified the Python conversion against their Matlab original with synthetic data. Further details of the uLSIF method are given in Appendix

B.3.

The uLSIF method has three parameters, the kernel size (*sigma*), the number of kernels (B) and the regularisation parameter (*lamdba*). The regularisation parameter was set automatically using cross validation within the uLSIF method.

Figure 5.5 shows how the distribution of weights varied with *sigma* and B for the four UCI_dev datasets..

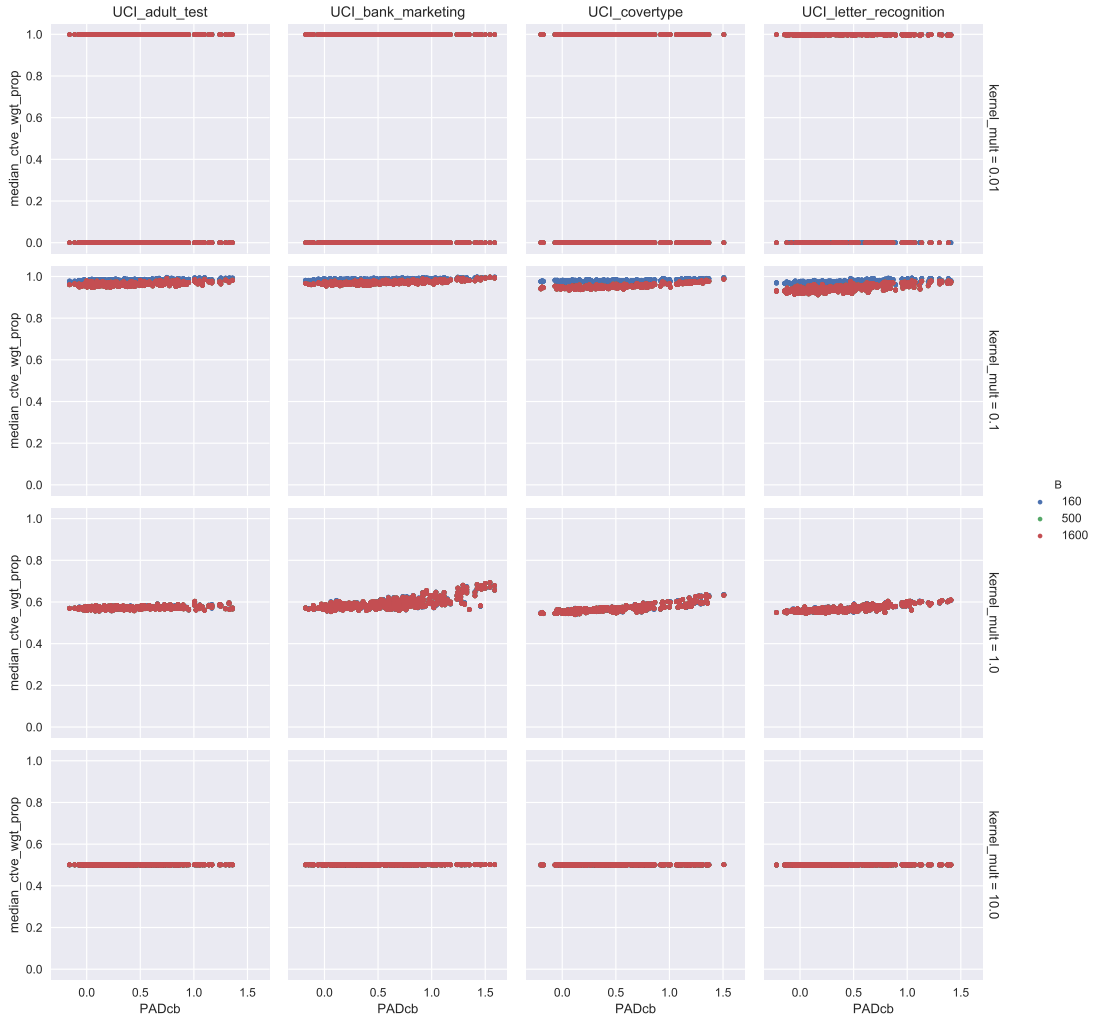

Figure 5.5: Median cumulative weight proportion from uLSIF method vs. level of dataset shift as measured by PADcb. Shown separately by dev dataset, kernel size multiple and B parameter

Again, a kernel size multiple of 0.01 gives an unreliable allocation of weights. Weight allocation is broadly consistent across the datasets. With a kernel size multiple of 0.1 most of the weight is allocated to a small number of instances while at a size multiple of 10.0 weight is allocated evenly across instances. Weight distribution does not vary as

much with the level of dataset shift as it does with the KMM method.

5.6 Instance weights by class

Instance weighting methods compute an importance weight for each instance in the training set that minimises the difference between the weighted distribution of the training set and the distribution in the test set. The aim of this section was to explore the relationship between the distribution of instance weights by class and the class distribution of the test set.

5.6.1 Method

Training, validation and biassed test sets were repeatedly sampled from the UCI datasets. Weights for the instances in the training set were generated with both the uLSIF and KMM methods.

5.6.1.1 Algorithm

Data: UCI Datasets (7)
Result: Instance weights by class
for *UCI dataset* **do**
  Find optimal classifier parameters for the dataset;
  for *Number of iterations* **do**
    Sample an unbiassed training set from the UCI dataset;
    Sample an unbiassed validation set from the UCI dataset;
    Sample a *biassed* test set from the UCI dataset;
    for *KMM methods and parameter settings* **do**
      Compute instance weights;
    end
    for *uLSIF methods and parameter settings* **do**
      Compute instance weights;
    end
  end
  Write results to file;
end

Algorithm 5: Instance weighting by class

5.6.1.2 Parameter settings

Table 5.1: Overall parameter settings

Parameter	Value	Notes
Training set size	1600	
Training set class 0 proportion	0.5	
Test set size	500	
Test set class 0 proportion	[0.4, 0.96]	See footnote[2]
Classifier type	SVM	
Classifier kernel	{Linear, RBF}	Optimised
Classifier hyper-parameters		Optimised[3]

Table 5.2: KMM parameter settings

Parameter	Value	Notes
Kernel size multiples[4]	{0.01, 0.1, 1.0, 10.0}	Optimum selected
Number of kernels (B)	{160, 500, 1600}	All

Table 5.3: uLSIF parameter settings

Parameter	Value	Notes
Kernel size multiples	{0.01, 0.1, 1.0, 10.0}	Optimum selected
Number of kernels (B)	{160, 500, 1600}	All
Regularisation parameter (*Lambda*)		Set automatically

Figures 5.4 and 5.5 showed previously that the results were insensitive to the B parameter so the results from all B parameter settings were used.

Figure 5.6 shows the accuracy achieved with the selected parameters for each dataset. Parameters were set only once per dataset but the program was run multiple times hence the distribution of accuracy values for each dataset.

[2]To minimise the impact of clipping while still retaining a wide range of class proportions

[3]The kernel and hyper-parameters were selected that gave the highest accuracy on the validation set

[4]*sigma* equals the median instance spacing for the dataset multiplied by the kernel size multiple

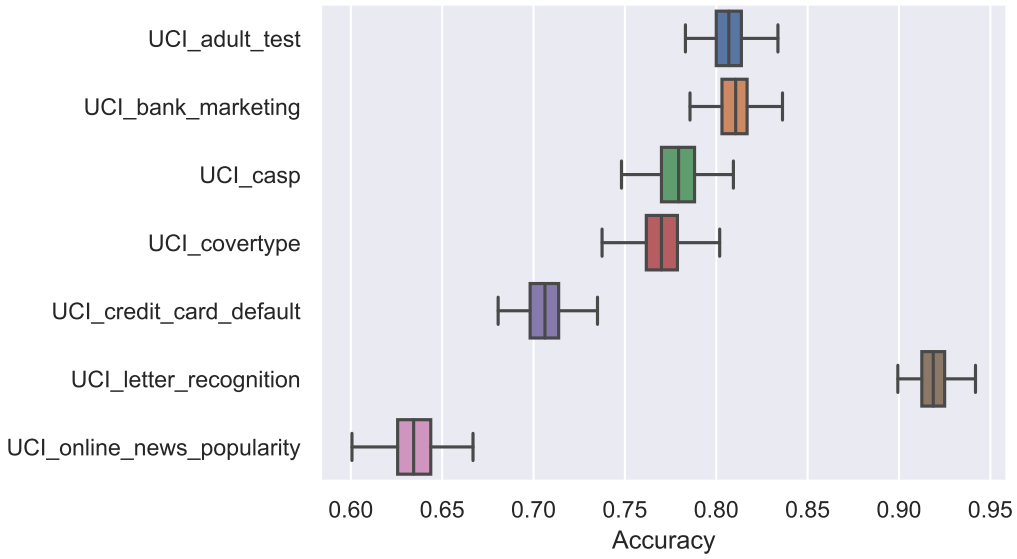

Figure 5.6: Classifier accuracy by dataset

The accuracy figures are clearly better than those obtained on the same datasets in Chapter 4 as shown in Figure 4.5. In part this will be due to the size of the training set (1,600 instances vs. between 100 and 10,000). However, a large part of the improved accuracy is likely to be due to a wider choice of classifier tuning parameters. Classifier parameters are set on each run for each dataset. Table 5.4 shows the proportion of times that the selected kernel was linear or RBF. It shows that in the majority of cases a Radial Basis Function (RBF) kernel gave the highest classifier accuracy. In Chapter 4, only linear kernels were used.

Table 5.4: Classifier kernel selection by dataset

Dataset	Linear	RBF
UCI_adult	0.76	0.24
UCI_bank_marketing	0.15	0.85
UCI_casp	-	1.0
UCI_covertype	-	1.0
UCI_credit_card_default	0.46	0.54
UCI_letter_recognition	-	1.0
UCI_online_news_popularity	0.29	0.71
Overall	0.32	0.68

The rbf kernel has two parameters while the linear kernel has just one. Including the rbf kernel considerably increased the size of the parameter space over which to search. Accuracy was measured on a total of 55 parameter combinations. To keep run times down, this parameter tuning was done once per dataset per run. Slightly higher accuracies may have been obtained by tuning the parameters for every sampled training set but it was felt that any improvement would be marginal and not worth the impact on run time.

As discussed in Section 2.3.2.3, the results for the KMM method applied to classification reported in Gretton et al. [58] were not as good as those reported earlier by the same authors in Huang et al. [66]. Gretton et al. [58] put this down to fitting over simple hypotheses in the earlier work. The intention of optimising the classifier over a wide range of hyper-parameter settings was to avoid similar problems and to avoid crediting the KMM-based quantification with improvement that could be obtained by simply fitting a better classifier.

5.6.2 Results

Figures 5.7 and 5.8 show the relationship between the proportion of total instance weight by class against the class distribution of the test set, with instance weights computed by KMM and uLSIF respectively. Colours identify the degree of dataset shift between the training and test sets with quartile 1 representing the lowest level of dataset shift and quartile 4 representing the highest dataset shift as measured using PADcb.

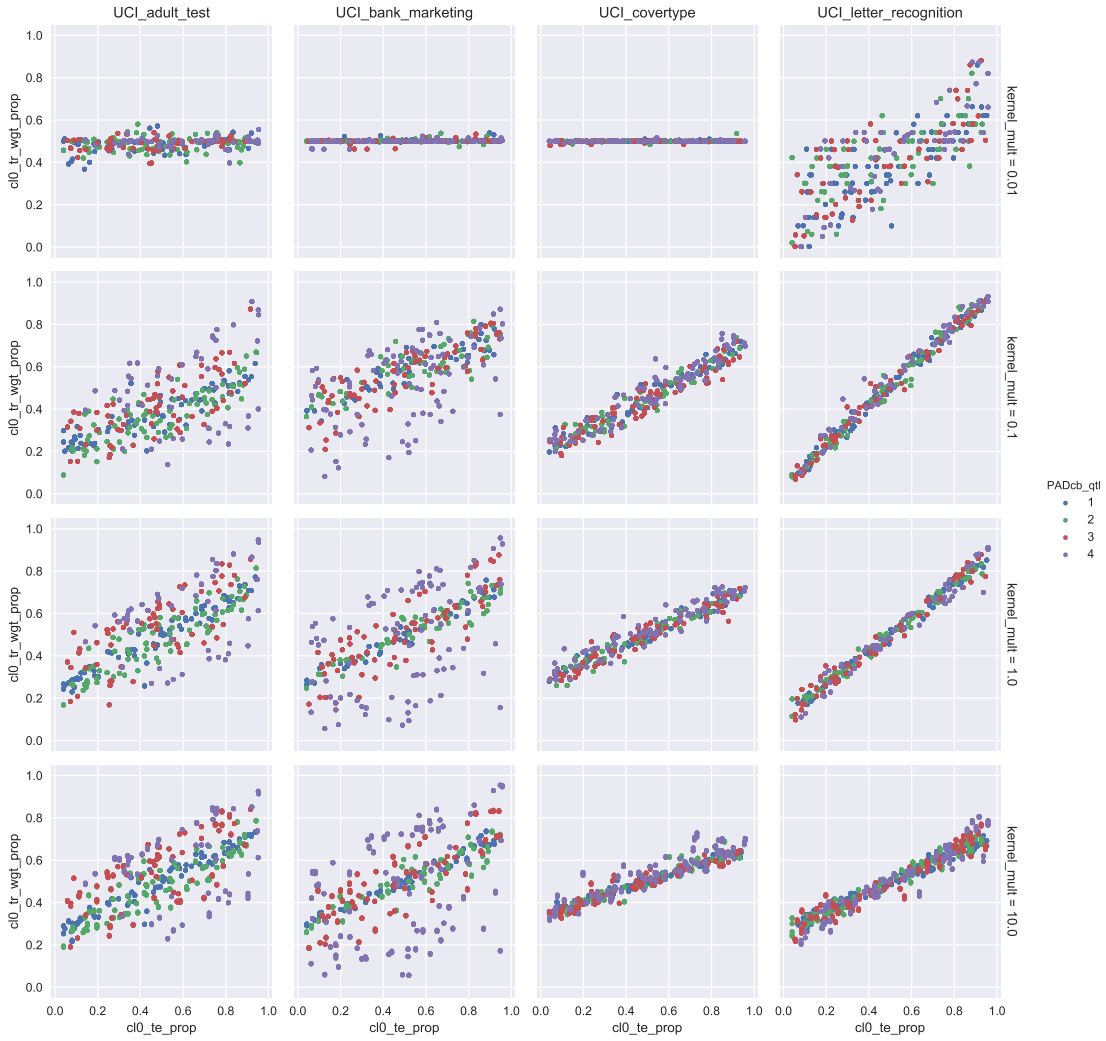

Figure 5.7: Proportion of instance weight applied to class 0 instances in the training set vs. proportion of class 0 instances in the test set. KMM method. UCI_dev datasets. Kernel size multiples {0.01, 0.1, 1, 10}

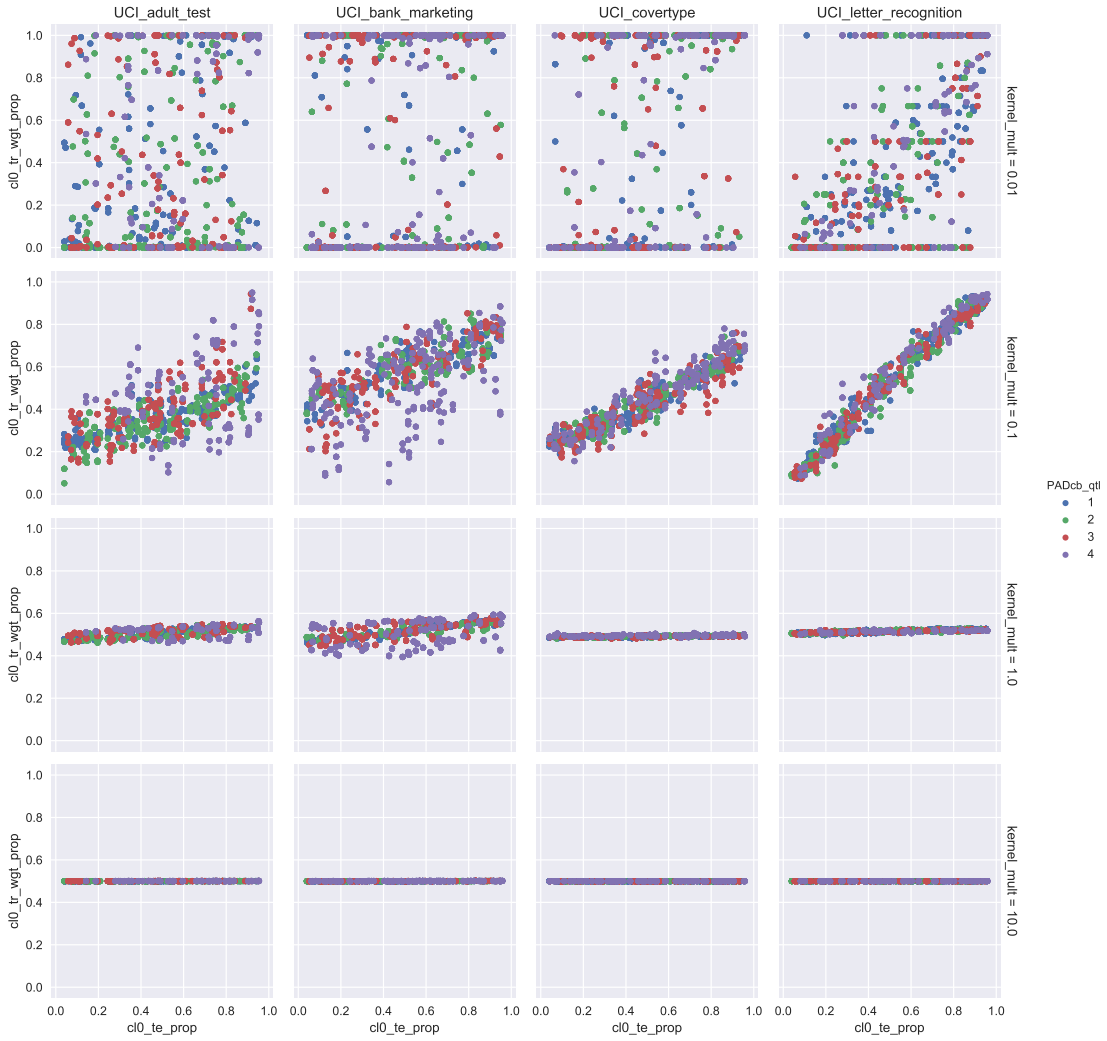

Figure 5.8: Proportion of instance weight applied to class 0 instances in the training set vs. proportion of class 0 instances in the test set. UCI_dev datasets. uLSIF method. Sigma parameter {1.0, 3.162, 10.0}

5.6.3 Discussion

Figures 5.7 and 5.8 show that in some cases there is a positive correlation between the proportion of the total weight assigned to a class by the instance weighting method and the proportion of that class in the test set. A positive correlation indicates that some of the instance weight is as a result of a difference in $P(y)$, the class distribution, between the test set and the training set.

5.7 Quantification by instance-weighted classify and adjust (IWCA)

In this section, the weighted instances are used in combination with a *classify and adjust* quantifier (Section 2.1.2). The hypothesis is that if the class-conditional feature distribution of the weighted training set is close to that of the test set then a *classify and adjust* quantifier trained with the *weighted* training set will give a good quantification performance on the test set.

However, these instance weighting methods look to minimise the *overall* distance between the distribution of the training set and the distribution of the test set. As discussed earlier, this difference can be disaggregated into a difference in class distribution $P(y)$ which is to be expected in quantification, and a difference in class-conditional feature distribution $P(x|y)$ which we would like to reduce or eliminate if a standard *classify and adjust* quantifier is to work effectively. The results in Section 5.6 showed that instance weights do sometimes correlate with class proportions.

There are several approaches that can be taken to try to focus the instance weighting on addressing differences in class-conditional feature distribution while ignoring differences due to class distribution. These are set out in Table 5.5.

Table 5.5: Approaches to introduce class-conditionality into instance weighting

1	**Class-balanced weights**	Scaling the instance weights of each class to ensure that the ratio of total sum weight assigned to instances of each class is equal to the ratio of the number of instances of each class
2	**Class-balanced thresholding**	Using instance weights as an indicator of which instances to keep and which to remove from the training set and removing equal proportions from each class
3	**Matched sub-samples**	Sub-sampling the training set to sets of the same class proportions as the estimated class proportions of the test set. Averaging instance weights for repeated sub-samples from the training set until all instances in the training set have been weighted
4	**Weakly supervised**	Identifying instances in the test set with the strongest likelihood of membership of a particular class. Using these instances to compute weights for the training set separately for each class.

Approaches 1, *class-balanced weights*, and 2, *class-balanced thresholding*, are used in the experiment in Section 5.7.1. Approach 3, *matched sub-samples* is applied in Section 5.9. A range of experiments were carried out with the *weakly supervised* approach but the results were no better than the baseline.

5.7.1 Method

The underlying method was the same as that set out in Section 5.6.1, in fact the same experiment generated the data for both sections. Instance weighting was made both with the KMM and uLSIF methods. Quantification was made with a matrix-inversion *classify and adjust* method (see Section 2.1.2.1). As is usual with matrix-inversion methods, the class proportion estimates were clipped to lie within [0,1].

The computed instance weights were either used directly as instance weights (**'w'**) or for selecting which instances to keep and which to remove from the respective set (**'t'**).

Alternatively the weights were not used and the instances simply remained unweighted (**'u'**).

When the weights were used directly (**'w'**) they were used *both* as-computed by the instance weighting method *and* balanced by class as described in approach 1, *class-balanced weights*, in Table 5.5.

When the weights were used in a thresholded way (**'t'**) to identify which instances to keep in the set and which to remove, the same proportion of instances *in each class* with the highest weights were retained. The others were removed. This is approach 2, *class-balanced thresholding*, in Table 5.5.

The matrix-inversion method relies on two sets of labelled data: a *training set* to train the classifier and a *validation set* on which to compute the values of classifier recall from which the adjustments are calculated. To make maximum use of the available labelled data, a deployed version would probably utilise cross-validation instead of a separate validation dataset. However, in this experiment there is a plentiful supply of labelled data and using separate training and validation sets was quicker than cross-validation.

Five overall methods were created by making logical combinations of the three approaches (**u**, **w** and **t**) for training the classifier and for computing the recall values. These are shown in Table 5.6.

Table 5.6: Methods used to combine computed instance weights with the matrix-inversion quantification method

Method	Classifier Training	Computing Recall	Note
	tr set	val set	
uu	unweighted	unweighted	baseline
ut	unweighted	thresholded	
uw	unweighted	weighted	
tt	thresholded	thresholded	
ww	weighted	weighted	

The **uu** method, which takes no account of instance weighting, is the *baseline* against which the other methods are measured.

On each iteration a training, validation and test set was sampled from the full dataset. In total there were 259 iterations for each dataset. The KMM and uLSIF methods were

applied on each iteration, each with a range of parameters. In total there were 480 method-parameter combinations applied to each of the 259 iterations of each dataset.

With a wide range of biassing and parameter settings, occasionally the matrix of recall values could not be inverted. When this happened the baseline **uu** result was returned.

5.7.2 Results

5.7.2.1 Ranked comparison on the UCI_dev datasets

The method-parameter settings were ranked by the absolute error between the estimated class distribution and the true value of the class distribution for each of the dataset-iterations. The mean value of both the absolute error and the ranking was computed across all the dataset-iterations.

5.7.2.2 Friedman test with Bonferroni corrections on the UCI_dev datasets

The statistical significance of the results from each method-parameter combination was tested using a Friedman test with Bonferroni corrections [2][31]. This test identified the method-parameter settings where the null-hypothesis of no difference to the baseline **uu** method could be rejected with an α of 0.05. All of the method-parameter settings with lower MAE than the baseline **uu** method that passed this test are given in Table 5.7 along with the **uu** baseline.

Table 5.7: Method-parameter settings with lower MAE than baseline which are statistically significant at $\alpha < 0.05$ under the Friedman test with Bonferroni corrections. UCI_dev datasets.

Method	Weighting	Class Balanced	Threshold Value	Number of Kernels	Kernel Multiple	MAE	Mean Rank
uu						0.100	128.0
ut	KMM		0.5	160	1.0	0.088	106.7
ut	KMM		0.5	1600	1.0	0.088	107.0
ut	KMM		0.5	500	1.0	0.088	107.1
ut	KMM		0.7	500	1.0	0.091	108.9
ut	KMM		0.7	160	1.0	0.091	109.1
ut	KMM		0.7	1600	1.0	0.091	109.2
ut	KMM		0.7	160	0.1	0.092	111.0
ut	KMM		0.7	500	0.1	0.092	111.0
ut	KMM		0.7	1600	0.1	0.093	111.2
ut	KMM		0.9	160	10.0	0.094	112.6
ut	KMM		0.9	1600	10.0	0.095	112.7
ut	KMM		0.9	500	10.0	0.095	113.0
ww	uLSIF	True		160	1.0	0.095	113.8
ut	KMM		0.9	1600	1.0	0.096	114.1
ut	KMM		0.9	160	1.0	0.096	114.2
ww	uLSIF	True		500	1.0	0.095	114.2
ww	uLSIF	True		1600	1.0	0.095	114.2
ut	KMM		0.9	500	1.0	0.096	114.4

From the results in Table 5.7 the best method is:

Table 5.8: Best method by mean rank from Table 5.7

Instance weighting:	KMM with a kernel size equal to the median L_2-norm distance between instances in the dataset
Classifier training:	Full training set, unweighted
Recall calculation:	The 50% of the instances in the training set with the highest weights, then equally weighted

The best method used 160 kernels but as can be seen from Table 5.7 the methods using 1600 and 500 kernels had very similar performance. A formal test of statistical significance between these methods was not carried out but, given the similarity of the results, it is unlikely that the number of kernels is significant.

5.7.2.3 Class-conditionality

All of the methods in Table 5.7 have some measure of class-balancing. Looking back to Table 5.5, the **ut** method is *class-balanced thresholding* while the **ww** methods were those that used *class-balanced weights*.

5.7.2.4 Wilcoxon signed-rank test on the UCI_test datasets

Having established a best method against the UCI_dev datasets, this method was compared to the **uu** baseline method on the three held-out UCI_test datasets. The p-value for the null hypothesis that the results from this method were no different to the results from the baseline method was computed using a Wilcoxon signed rank test [31][120]. The results are shown in Table 5.9

Table 5.9: Performance of best method from UCI_dev datasets on the held-out UCI_test datasets

Method	Weighting	Threshold Value	Number of Kernels	Kernel Multiple	MAE	Mean Rank	p-value
uu baseline					0.122	1.55	
ut	KMM	0.5	160	1.0	0.109	1.45	1.7e-4
ΔMAE	absolute				0.013		
ΔMAE	relative				10.7%		

On the on the UCI_test datasets, the method selected as best using the results on the UCI_dev datasets gave an absolute improvement in MAE of 1.3 percentage points against the **uu** baseline method, a relative improvement of 10.7%. With a p-value of 1.7e-4 the null-hypothesis can be rejected with a high degree of confidence.

5.7.2.5 Performance vs. dataset shift

Figures 5.9 and 5.10 show the MAE of the best method from Table 5.8 and the **uu** method baseline against level of dataset shift for the UCU_dev and UCI_test groups of datasets respectively. Figure 5.11 shows the difference in MAE between the the best method and the baseline, for each dataset individually.

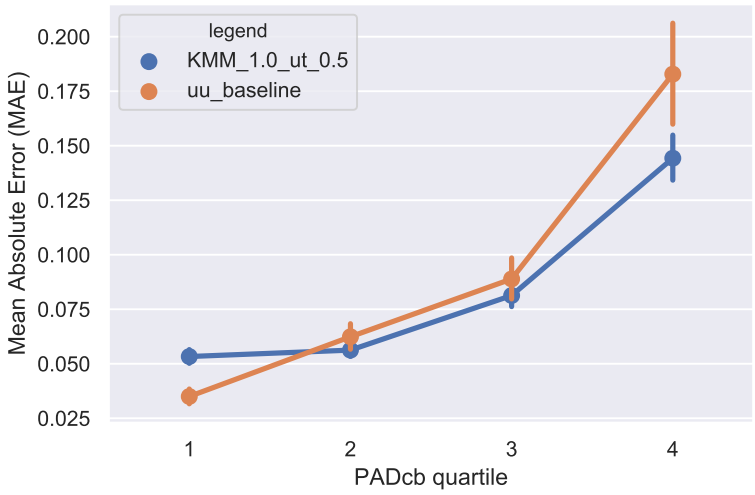

Figure 5.9: MAE of ut method with KMM weighting, threshold=0.5, kernel size multiple=1.0 and uu baseline method vs. PADcb quartile. UCI_dev datasets. 95% confidence intervals

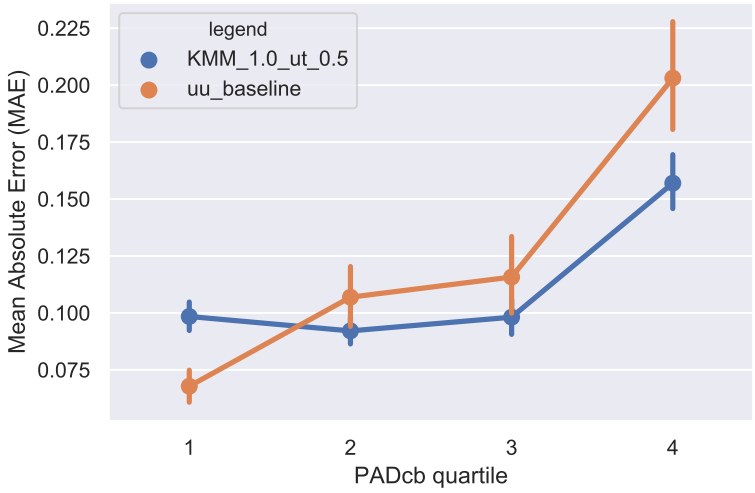

Figure 5.10: MAE of ut method with KMM weighting, threshold=0.5, kernel size multiple=1.0 and uu baseline method vs. PADcb quartile. UCI_test datasets. 95% confidence intervals

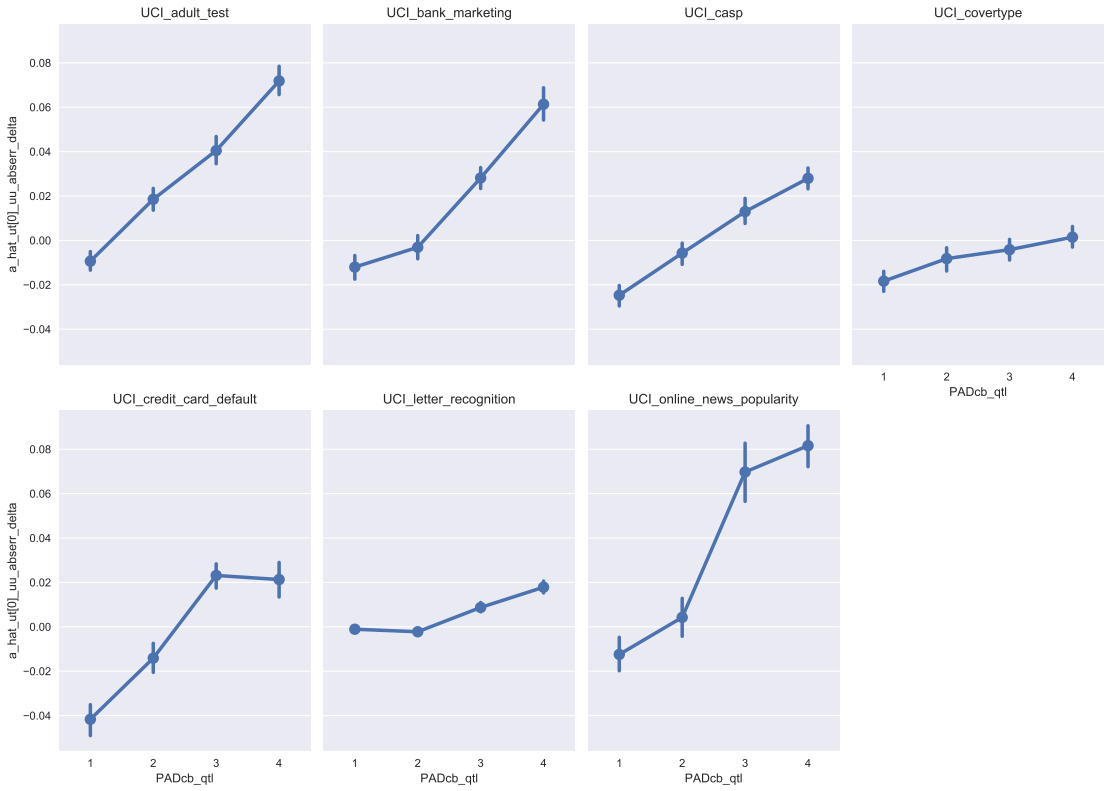

Figure 5.11: Delta MAE of ut method with KMM weighting, threshold=0.5, kernel size multiple=1.0. vs uu baseline method by PADcb quartile. 95% confidence intervals

The methods have been evaluated over a set of test sets. The distribution of dataset shift of those test sets has been shown earlier in Figures 5.2, 5.3 and 5.1. Figure 5.11 shows that different methods have different levels of performance relative to the baseline at different values of class-conditional dataset shift. Clearly, a different distribution of class-conditional dataset shift may give a different overall result. The best method will depend on the expected distribution of class-conditional dataset shift. This is discussed further in Section 7.1.

For example, Table 5.7 shows that when the threshold value is increased from 0.5 to 0.7, overall performance over the full range of test sets drops slightly. However, as shown in Figures 5.12 and 5.13, performance relative to the baseline improves when bias is low.

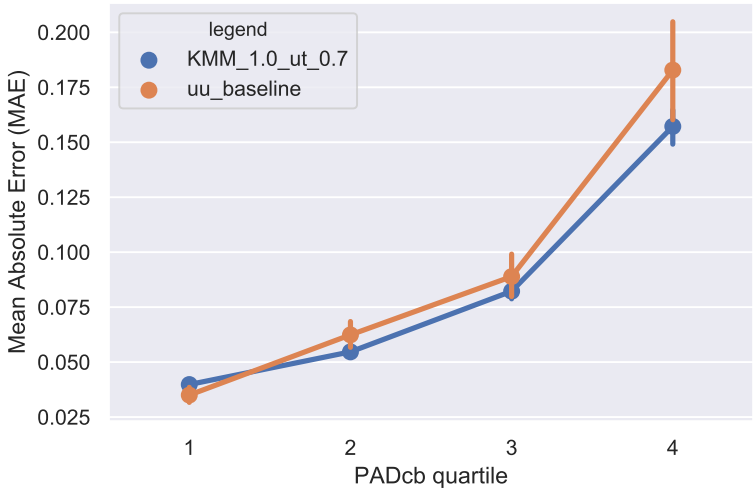

Figure 5.12: MAE of ut method with KMM weighting, threshold=0.7, kernel size multiple=1.0. and uu baseline method vs. PADcb quartile. UCI_dev datasets. 95% confidence intervals

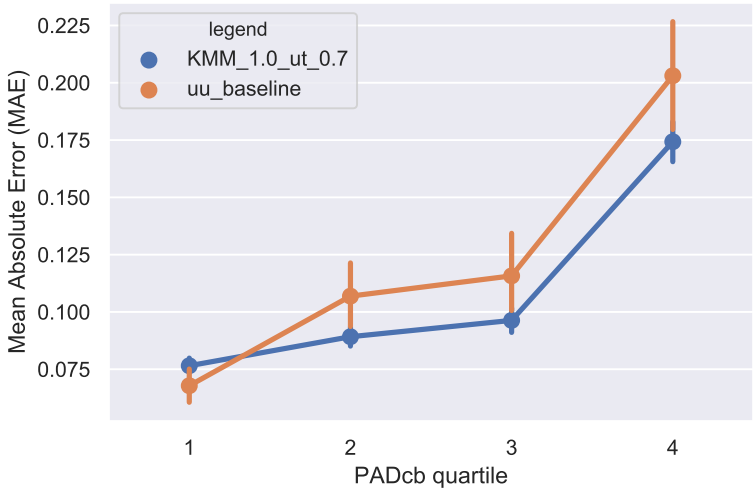

Figure 5.13: MAE of ut method with KMM weighting, threshold=0.7, kernel size multiple=1.0. and uu baseline method vs. PADcb quartile. UCI_test datasets. 95% confidence intervals

This method is more conservative. The level of potential improvement over the baseline is lower but so is the scale of any possible performance drop at low levels of dataset shift.

5.7.2.6 Comparative performance of methods

Table 5.10 is taken from the same source as Table 5.7 above and shows the settings that achieved the lowest MAE on the UCI_dev datasets for each of the five methods listed in Table 5.6 for each of the two weighting methods (KMM and uLSIF).

Table 5.10: Best method-parameter by MAE on UCI_dev datasets for both KMM and uLSIF instance weighting

	Weights	Class Bal	Thr Value	Num Kernels	Kernel Mult	Mean Rank	MAE	ΔMAE Abs.	ΔMAE Rel.
uu						128.0	0.100		
tt	KMM		0.5	1600	1.0	116.1	0.091	0.009	9.0%
tt	uLSIF		0.9	500	0.1	123.4	0.099	0.001	1.0%
ut	KMM		0.5	160	1.0	**106.7**	0.088	0.012	12.0%
ut	uLSIF		0.7	160	1.0	123.2	0.096	0.004	4.0%
uw	KMM	False		1600	0.01	133.4	0.104	-	-
uw	uLSIF	False		160	1.0	116.5	0.097	0.003	3.0%
ww	KMM	True		1600	10.0	140.1	0.104	-	-
ww	uLSIF	True		1600	1.0	**114.2**	0.095	0.005	5.0%

Where the computed weights were then used with a thresholded approach (i.e. the **ut** and **tt** methods) the weights computed by KMM gave better results than those computed by uLSIF. The uLSIF method gave better MAE than KMM with the **uw** and **ww** methods.

The thresholded methods (**ut** and **tt**) with KMM instance weighting delivered lower MAE than the methods that used the weights directly. However, only the ut-KMM and ww-uLSIF methods were statistically significantly different from the baseline under the Friedman-Bonferroni test with an α of 0.05. These are indicated in bold in Table 5.10.

5.7.3 Discussion

IWCA appears to be a good method of improving quantification accuracy under conditions of class-conditional dataset shift. The method-parameter setting that gave the best overall quantification accuracy used KMM-computed weights in a thresholded way to select the 50% of the instances in the validation set that was had been assigned the highest instance weights. *All* of the training set was used, unweighted, to train the classifier. The best parameter settings gave an absolute improvement in mean absolute quantification error of 1.3% over the baseline, a relative improvement of 11%.

Class-conditionality appears to be important, with *all* of the methods that were significantly better than the baseline being class-balanced in some way.

5.8 Classifier-based sample selection bias correction (SSBC)

The Zadrozny [122] method for *instance weighting*, Sample Selection Bias Correction (SSBC), pre-dates the KMM and uLSIF methods by a number of years and other domain adaptation methods have usually been shown to be superior. However, the results in Section 5.7.2 showed a quantification method based on the older KMM method out-performed a method based on the newer uLSIF method. Given this result, I was interested to see if the potentially inferior SSBC method might actually out-perform both of these methods when applied to the quantification task.

In SSBC the training set is considered to be a sample drawn from the Target domain with a sampling bias. Whether an instance is in the training set or not can be thought of as being determined by the value of a selector variable s. Instances that are in the training set have a selector variable value of 1, those not in the training set have a selector variable value of 0. Zadrozny [122] makes the assumption that the probability that the value of the selector variable being 1 is a function of x *only* and is independent of the class label y i.e. $P(s = 1|x, y) = P(s = 1|x)$.

To correct for this sample selection bias the weight of each instance i, w_i, is given by:

$$w_i = \frac{P(s = 1)}{P(s = 1|x_i)}, \tag{5.2}$$

where $P(s = 1)$ is the overall selection probability:

$$P(s = 1) = \sum_{(x,y,s)\sim T} P(s = 1, x). \tag{5.3}$$

The values for $P(s = 1|x)$ for the instances in the training set are estimated by training a probabilistic classifier (in this case logistic regression) with a dataset containing instances from the test set labelled as s=0 and instances from the training set labelled as s=1.

5.8.1 Method

The method of quantification using instance weights was the same as set out previously in Section 5.7. If the test set was smaller than the training set then over-sampling with replacement was used to bring it up to the same size.

The method for computing the instance weights used the sci-kit learn LogisticRegressionCV classifier with a linear kernel. This classifier optimised its only hyper-parameter,

the regularisation parameter, on each iteration using cross validation with the aim of maximising classification accuracy. As before, in each iteration a training, validation and a biassed test set was sampled from the full dataset. 56 iterations were made for each dataset. While the computation of instance weights did not require any external parameters, there were 20 different parameter settings for each iteration that controlled the application of those weights to quantification. The parameter settings that gave the best quantification performance are shown below.

5.8.2 Results

The table below gives the three best parameter settings as measured by mean-rank, MAE and RMSE respectively for the UCI_dev datasets. The best figures for each are shown in bold. The p-values are computed based on the MAE values for the chosen method vs. those from the baseline **uu** method across the 224 iterations (56 iterations x 4 datasets).

Table 5.11: Best SSBC methods by mean rank, RMSE and MAE. UCI_dev datasets

Method	Class Bal.	Thr. Value	Mean Rank	RMSE	MAE	ΔMAE Abs.	ΔMAE Rel.	p-value
uu			10.3	0.192	0.121			
ww	True		**9.2**	0.178	0.123	-	-	
ut	True	0.7	9.5	**0.169**	0.128	-	-	
ut	True	0.9	9.4	0.179	**0.119**	0.002	1.7%	0.107

As before in Section 5.7, a Friedman test with Bonferroni corrections [2][31] was carried out on these results. Only the parameter setting that gave the best MAE had a lower MAE than the baseline but its p-value was far below the level at which the null hypothesis could comfortably be rejected[5].

These three best methods were compared to the baseline **uu** method on the UCI_test datasets.

[5] an α of 0.05 equates to a p-value of below 0.0025

Table 5.12: Best SSBC methods by mean rank, MAE and RMSE. UCI_test datasets

Method	Class Balanced	Threshold Value	Mean Rank	MAE	RMSE
uu baseline			8.6	0.107	0.166
ww	True		9.8	0.146	0.217
ut	True	0.9	8.0	0.108	0.164
ut	True	0.7	9.1	0.131	0.181

On the held-out UCI_test data *none* of the methods identified as best from the UCI_dev data had an MAE lower than the baseline.

5.8.3 Discussion

From these experiments, a quantifier based on the Zadrozny [122] SSBC method is inferior to the KMM and uLSIF based methods in Section 5.7.

A linear kernel was used for the classifier which computed instance weights. A comparison of Figures 4.5 and 5.6 indicates that with some of the datasets the Radial Basis Function (RBF) kernel can give higher classification accuracy. It may be that using a classifier with an RBF kernel in the SSBC method would have given a better quantification performance.

5.9 Iterative test-train bias reduction (ITTBR)

Section 5.7 showed that class-balanced methods that used the instance weights computed by the KMM method could successfully increase quantification accuracy under class-conditional dataset shift.

The approach taken in this approach was iterative and based on the SSBC concept of using a classifier to weight instances in the training set by their closeness to the instances in the test set as we applied in Section 5.8. The process was:

- The estimate of class proportions in the test set is estimated using a *classify and adjust* quantifier that is trained on all the currently active training data

- Multiple samples of this estimated class proportion are drawn from the currently active training set and with each sample a probabilistic classifier is trained to distinguish between instances from this training set sample and the test set.
- A fixed proportion of the currently active training set with the lowest probability of being from the test set (averaged over all of the samples) are removed from the currently active training set
- The process repeats with the new currently active training set

The rationale was that with an iterative approach as the least relevant training instances were removed the quantifier becomes more accurate; and so the class-balance of the sample from the training set becomes more accurate; and so the identification of the least relevant instances from the training set becomes more accurate.

Looking at the baseline results from the UCI_dev datasets (Figure 5.14) supports the argument that when the training and test datasets are close, as measured by PADcb, then the accuracy of a standard *classify and adjust* quantifier is higher.

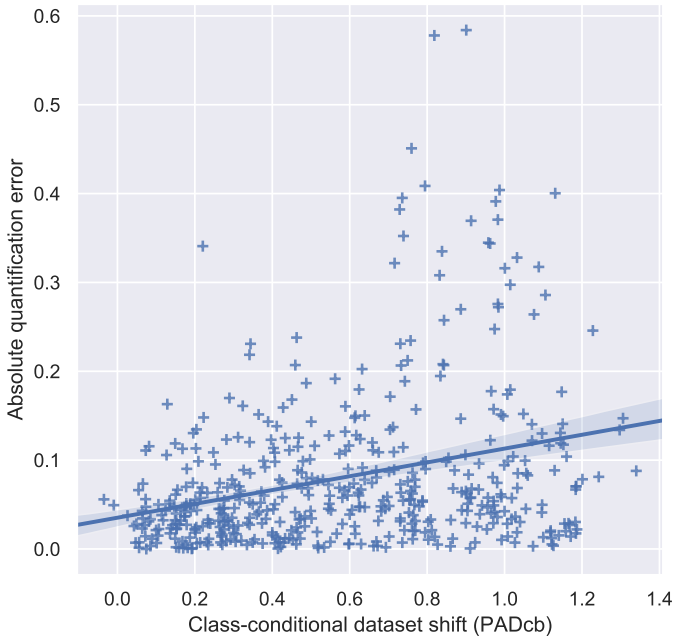

Figure 5.14: Absolute quantification error using classify and adjust method vs. class-conditional dataset shift.

The hypothesis that is tested in this section is that by *actively reducing* the distance

between the training and test sets as measured by PADcb then quantification accuracy will *improve*.

5.9.1 Method

A logistic regression classifier (clf_trte) was used to discriminate between instances from the training set and instances from the test set. The classifier was trained on a training set made up of the whole of the test set (labelled 'te') and a similarly sized sample from the active training set (labelled 'tr'). To minimise the potential impact of class imbalance the class proportion of the sampled active training dataset was set to be the same as the estimated class proportion of the test dataset. This was previously identified as approach 3 in Table 5.5. This class proportion is itself estimated using a second classifier trained on the active training set but using the class labels (clf_01). The class proportion in the test set was then estimated using the Saerens EM method (see Section 2.1.2.2). As there will be class-conditional dataset shift, this method will not give a perfect estimate of the class proportions. This is the rationale behind the iterative approach. By only removing the training instances that are furthest from the test set on each iteration, the estimate of the class proportions should improve and the identification of the training instances furthest from the test set should also improve. The Saeren's EM method requires a classifier that gives probabilistic outputs. A Logistic Regression classifier was used in this experiment.

The clf_trte then assigns tr-te probabilities to the instances in the active training set that have not been used in training clf_trte. By repeated sampling of the training set for clf_trte and averaging of the instance tr-te probabilities, eventually a sufficiently large proportion of the active training set will have an estimated tr-te probability. At this point a fixed proportion of the active training set with the lowest probability of being in the test set are removed from the active training set. The process then repeats.

The same seven UCI datasets were used. As before, the training and biassed test sets were sampled from full UCI dataset. This was done 30 times per dataset. For each iteration of the training and test set there were 12 sub-iterations ('steps') in which the 15% of the currently active training set instances with the highest probability of being in the training set were removed from the active set.

Table 5.13: ITTBR parameter settings

Parameter	Value	Notes
Training set size	1600	See note[6]
Training set class 0 proportion	0.5	
Test set size	500	
Test set class 0 proportion	[0.4, 0.96]	See footnote[7]
Classifier type	Logistic Regression	
Classifier kernel	Linear	
Classifier C-value	$\{10^{-4}$ to $10^{4}\}$	Optimised[8]

[6]This was the initial size. The training set reduced in size by 15% on each iteration

[7]To minimise the impact of clipping while still retaining a wide range of class proportions

[8]The sklearn LogisticRegressionCV function was used which each time set the C-value through cross validation to be the value that maximised accuracy score

5.9.2 Algorithm

Data: UCI Datasets (7)

Result: Estimated test set class proportions

for *UCI dataset* **do**

  for *Number of iterations* **do**

    Randomly split into training and remainder;

    Sample a biassed test set from remainder;

    Set the training set as the active training set;

    for *Number of steps* **do**

      Train a classifier with the active training set;

      Use the classifier to estimate class probabilities of the instances in test;

      Apply Saerens EM to estimate class proportions in test;

      while *Proportion of active tr set* $<$ *target* **do**

        Sample a set from the active training set, same size as the test set and to estimated class proportions;

        Combine this training sub-set and the test set and train a classifier with train-test labels;

        Use classifier to estimate tr-te probabilities of the other instances in the active training set;

        Compute running mean of tr-te probabilities for each instance in active training set;

        Compute proportion of active training set with non-null mean probability;

      end

      Compute PADcb;

      Remove fixed proportion of instances from active training set with highest mean probability of being from training set;

    end

  end

  Write results to file;

end

Algorithm 6: Iterative test-train bias reduction (ITTBR)

5.9.3 Results

Figure 5.15 shows that at each step the method reduces the distance between the training and test sets as measured by PADcb.

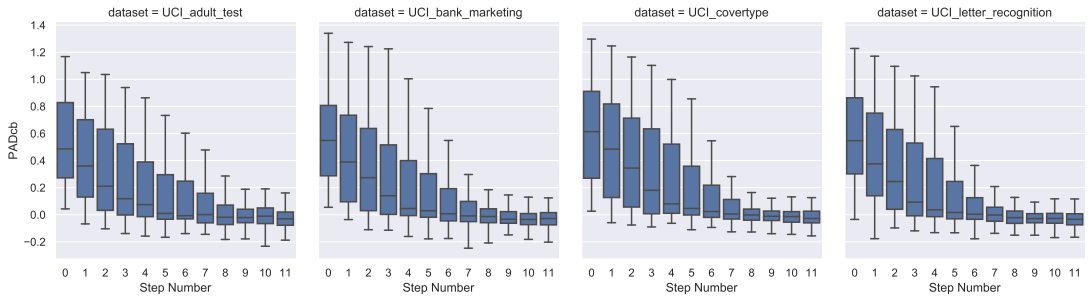

Figure 5.15: ITTBR. PADcb by sub-iteration step. UCI_dev datasets.

However, as shown below in Figure 5.16 on three of the four UCI_dev datasets, despite the reduction in PADcb the mean absolute quantification error vs. baseline (Delta MAE) worsened.

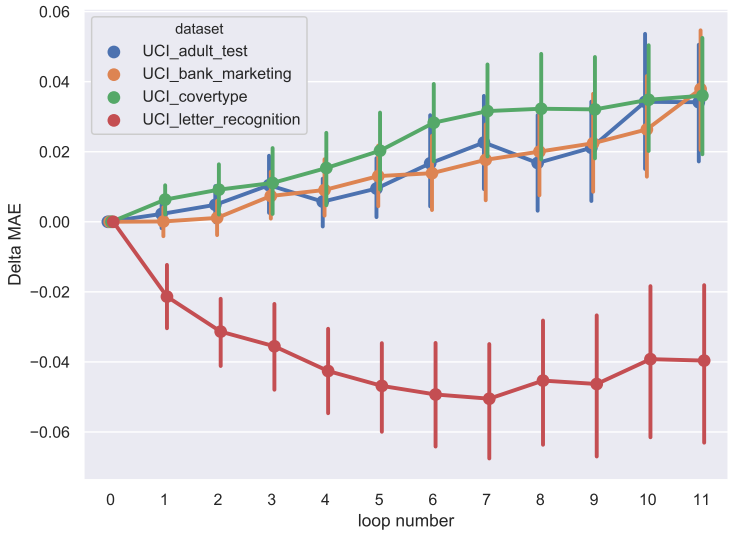

Figure 5.16: ITTBR. Absolute quantification error delta from initial value. Dev datasets. 95% confidence intervals

Some worsening of performance is to be expected as the size of the training set reduces by 15% on each step. Figure 5.17 shows the baseline effect of randomly removing 15% of the instances at each step.

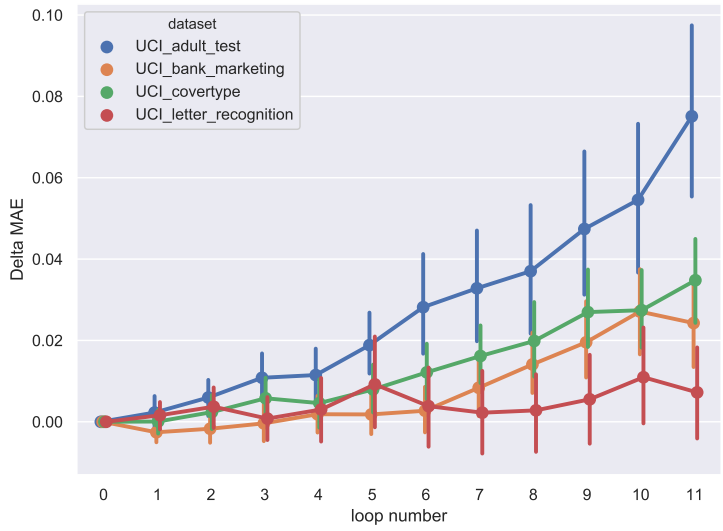

Figure 5.17: Baseline absolute quantification error delta from initial value. Points removed from training set at random. Dev datasets. 95% confidence intervals

Finally, Figure 5.18 is the error shown in Figure 5.16 *less* the random baseline error shown in Figure 5.17.

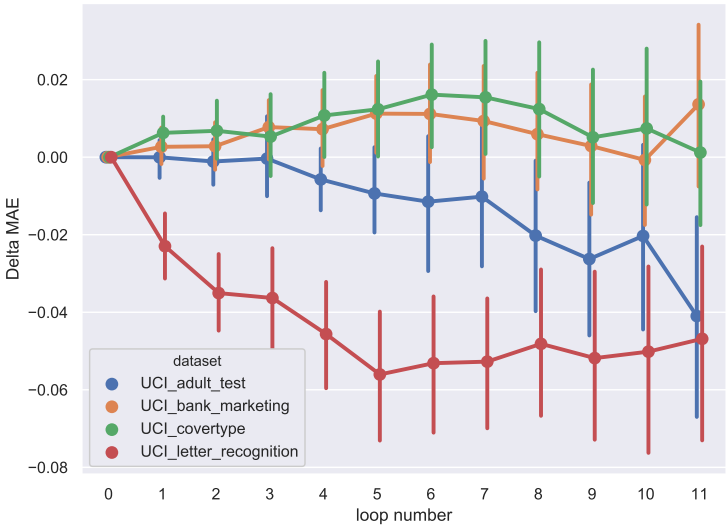

Figure 5.18: ITTBR absolute quantification error delta from initial value less baseline value. Dev datasets. 95% confidence intervals

On two of the four datasets, *UCI_adult_test* and *UCI_letter_recognition*, the ITTBR method improves on the baseline of removing datapoints at random. However, on the other two datasets, *UCI_bank_marketing* and *UCI_covertype*, it would be difficult to claim any significant difference between the ITTBR method and the random baseline.

Figure 5.19 shows the relationship between quantification accuracy and dataset shift between the active training set and the test set at different steps in the ITTBR process.

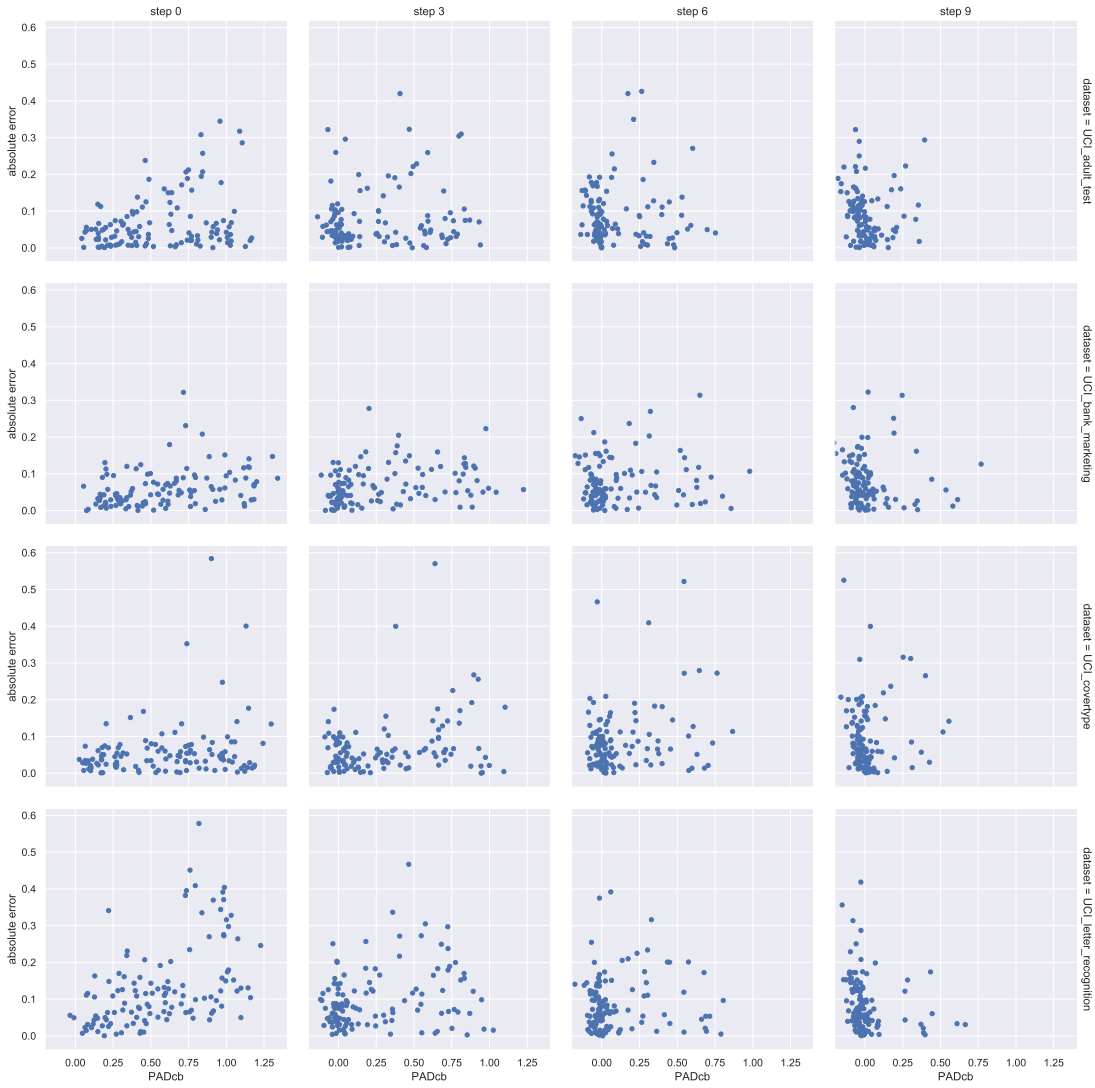

Figure 5.19: Absolute quantification error (actual vs. predicted class 0 proportion). Dev datasets.

The original hypothesis behind the ITTBR approach was motivated by the relationship between quantification accuracy and PADcb shown in Figure 5.14. If the relationship was causal from PADcb to quantification accuracy, then taking measures that reduce PADcb should improve quantification accuracy.

However, Figure 5.19 shows that while the ITTBR steps do reduce PADcb (points moving to the left on the x-axis) the quantification error (on the y-axis) does *not* reduce to the extent that had been hypothesised.

5.9.4 Discussion

The ITTBR method explicitly minimises the difference between the Source and Target domains as measured by PADcb. However, with a quantifier that relies on a classifier performing similarly in the two domains, quantification accuracy is poor. This appears to go against the theory espoused by Ben-David et al. [14]: 'Our theory ... also points toward a promising new model for domain adaptation: one which explicitly minimises the difference between the source and target domains, while at the same time maximising the margin of the training set.' Ganin and Lempitsky [46] also cited this in their work: 'Our approach is motivated by the theory on domain adaptation (Ben-David et al., 2006, 2010), that suggests that a good representation for cross-domain transfer is one for which an algorithm cannot learn to identify the domain of origin of the input observation.'

Chen et al. [29] and Glorot et al. [50] both found that the SDA architecture (see Chapter 6) improved domain adaptation but actually *increased* PAD. Again, an apparent contradiction to the theory of Ben-David et al. [14].

This throws up interesting questions for further work. As far as this thesis is concerned, the conclusion is that the ITTBR method of explicitly reducing PADcb may work well with certain datasets but overall gives poor quantification accuracy performance.

As with the SSBC method (Section 5.8), it could be that choosing a domain classifier with a higher accuracy (e.g. using an RBF kernel) would result in a better quantification performance. Again, another area for potential further work.

5.10 Conclusions

A summary of the results of the various experiments are given in Table 5.14.

[9] Was this finding statistically significant

Table 5.14: Summary of Chapter 5 results

		Method	Set On	Measured On	ΔMAE Absolute	ΔMAE Relative	Stat Sig[9]
Tab.	5.9	KMMut0.5	UCI_dev	UCI_test	1.3%	10.7%	yes
Tab.	5.10	uLSIFww	UCI_dev	UCI_dev	0.5%	5.0%	yes
Tab.	5.11	SSBCut0.9	UCI_dev	UCI_dev	0.2%	1.7%	no

A potential downside of instance weighting is that it effectively reduces the size of the training set. The negative impact on classifier performance of reducing the size of the training set is well documented (e.g. [41]) so clearly a balance needs to be struck between the potential positive and the potential negative effects of instance weighting. Interestingly, the best performing method utilises *all* of the training set for training the classifier but then calculates the classifier recall for the quantification adjustment using only the 50% of the validation data that is effectively closest to the data in the test set.

Another potential downside to importance-weighted quantification is run-time. The classifiers on which the quantification method is built cannot be trained until the instance weights are computed and the instance weights themselves cannot be computed until the test set is presented. Both of these two steps can be potentially time consuming and the instance weighting methods typically do not scale well to larger datasets. So, if fast run-times with large datasets are important, these methods could be problematic.

As shown in Figure 5.10, if there was *no* bias in the test set relative to the training set then even this best method would actually not perform as well as the baseline. In this chapter, 'best' has been judged on the basis of mean performance across the distribution of biassed datasets that was generated by the biassing method. In deployment the best method will depend on the expected level of bias. This is discussed further in Chapter 7.

The KMM, uLSIF and SSBC instance weighting methods are *not* inherently class balanced. *All* of the methods that had a quantification accuracy that was significantly better than the baseline had been separately class-balanced in some way.

Chapter 6

Domain adaptation with feature representations

In Chapter 5, instance weighting methods from domain adaptation were applied to the problem of quantification under class-conditional dataset shift. A method based on Kernel Mean Matching was found to be effective in increasing quantification accuracy. However, instance weighting methods for domain adaptation are not particularly new and methods that generate new feature representations are now often regarded as giving state-of-the-art performance. The objective of this chapter was to see whether a quantification method that was built on one of these feature representation approaches to domain adaptation would give better quantification performance than the methods that were built on instance weighting.

There are a variety of feature representation methods for domain adaptation. I chose the Stacked De-noising Autoencoder (SDA) method [50]. The SDA method appears to perform strongly and in its marginalised form [29] it is simple to implement and quick to run.

As in the previous chapters, we assume that the class-conditional feature distributions in the Source and Target domains are different, i.e. that $P_S(x|y) \neq P_T(x|y)$. With Stacked De-Noising Autoencoders the original features x are transformed into a new representation x'. A classifier is then trained on the new feature representation, x', to give an optimal classification hypothesis h. The inclusion of the hypothesis h is important. Glorot et al. [50] showed that the SDA approach to projecting the data into the new feature representation on its own did *not* reduce the distance between the Source and Target

domains. What it *did* do, however, was to disentangle information relating to domain from information relating to class. In finding an optimum hypothesis h a regularised classifier will give weight to the features that carry class information but not to those that only carry domain information. As such, it is reasonable to assume that $P_S(x'|y,h)$ and $P_T(x'|y,h)$ will become close. A standard *classify and adjust* quantifier can work effectively if $P_S(x'|y,h) \approx P_T(x'|y,h)$.

Quantification built on the Marginalised Stacked De-noising Autoencoder (mSDA) method [29] was incorporated into the same experimental approach that was used with instance weighting methods in Chapter 5. The results from both chapters are comparable.

6.1 Marginalised Stacked Denoising Autoencoders (mSDA)

An autoencoder can be thought of as a two-part device. The first part of the autoencoder, the *encoder* takes an input feature vector and transforms it. Very often (although not in the case of the mSDA) this transformation might be dimensionality reduction. The second part, the *decoder* is then the transformation of this *encoded* representation back to an output that is as close as possible to the original input. The encode and decode stages are typically trained by minimising a loss function on the accuracy of the re-creation of the original input. Instance labels are not needed. Training an autoencoder in this way is *unsupervised*. In a *denoising* autoencoder, the input values are partially corrupted with random noise. With the mSDA the noise takes the form of a set proportion of the input features being randomly set to zero.

In the SDA architecture the denoising autoencoders are *stacked* so that the output of the first denoising autoencoder is the input to the second, the output of the second is the input to the third etc. While Glorot et al. [50] used neural networks for the de-noising autoencoders, Chen et al. [29] found a way of using a simple and fast linear transformation as the denoising autoencoder layer.

Deploying a Stacked Denoising Autoencoder is a two step process. As a first step the SDA is trained using unlabelled instances from both the Source and the Target domains. The auto-encoders are trained greedily i.e. the first auto-encoder is trained from the data before the second is trained from the representation of the data generated by the first layer and so on. For a given input feature vector the new output feature vector is then

a concatenation of the outputs from the SDA layers. As a second step, the labelled data from the Source domain is processed by the SDA to give each instance the new feature representation. A classifier is then trained using the new feature representation and the original instance labels.

The literature around mSDAs and related methods is reviewed in Section 2.3.3.3

6.2 Method

The hypothesis at the core of this chapter is that with the new feature representations generated by the mSDA and the classifier hypothesis h, the Source and Target domains are closer than they were with the original features. Therefore the classifier should have less of a recall difference between the Source and Target domains than a classifier that was trained on the original features. With a smaller difference in recall between domains a *classify and adjust* quantifier should be more accurate in the Target domain than one trained on the original Source domain features.

6.2.1 Code

The Python code for the mSDA method was downloaded from Weinberger [119], a co-author[1] on Chen et al. [29].

6.2.2 mSDA training time

The Chen et al. [29] mSDA method is claimed to be significantly faster than the neural-network based SDA method from Glorot et al. [50]. However, Figure 6.1 shows that it still does not scale particularly well.

[1] Chen's PhD supervisor

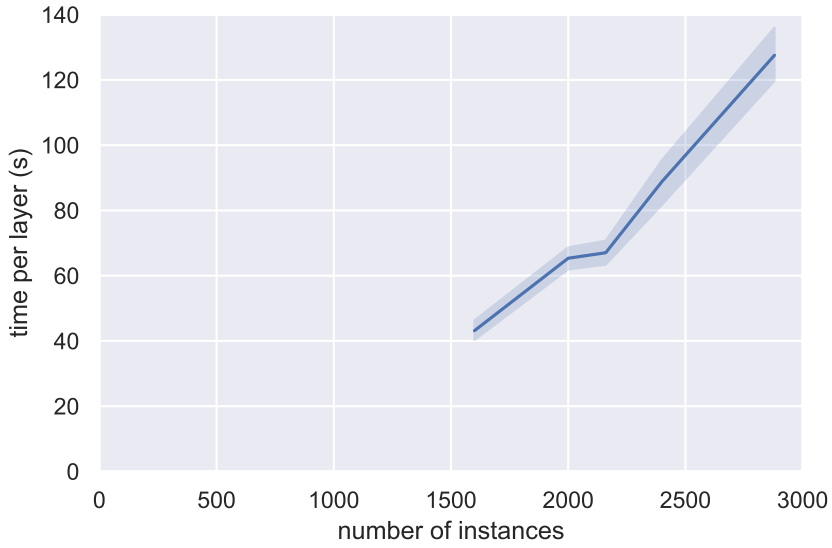

Figure 6.1: Mean mSDA calculation time per layer vs. number of data instances. 95% confidence intervals shown. All UCI datasets

6.2.3 Noise

The same approach was taken as that taken in the original paper by Chen et al. [29]. In each mSDA layer a fixed proportion of the input features are randomly set to zero with each data instance. In Chen et al. [29] the value for noise was determined by cross-validation. The approach taken here was to carry out quantification of the test set for all parameter combinations for later analysis. A value of 0.8 was chosen because Glorot et al. [50] had found that this value tended to be optimal. Chen et al. [29] had used 7 different noise values. To reduce the time required to run the experiment I used three values. A low value of 0.2, a medium value of 0.5 and the high value of 0.8.

6.2.4 Oversampling from the Target domain

It is not explicitly stated in Chen et al. [29] but the implication is that the datasets they used were all of equal size. In this experiment the training set was either 1200 or 1600 instances while the test set was either 400 or 800 instances. I hypothesised that the domain adaptation to the Target domain (from which the test set was drawn) will not be as good if the amount of training data for the mSDA from the Target domain is much less than the amount from the Source domain. To test this hypothesis additional Target domain data

was added to the set being used to train the mSDA. This additional data was generated by sampling with replacement from the test set. The amount of additional data that was added was such that such that the total number of instances from the Target domain would then be 80% of the number from the Source domain. As a baseline, no additional data was added.

6.2.5 Layers

Both Glorot et al. [50] and Chen et al. [29] limited their experiments to 5 SDA layers and I did the same. Table 6.1 shows how the outputs from the 5 layers plus original features (layer 0) were then concatenated in eleven different arrangements to form ten new feature representation (B to K) and the baseline representation of the original features (A).

Table 6.1: Ten feature representations plus baseline constructed from the original features (layer 0) and the five mSDA layers

A	0						baseline
B	0	1					
C	0	1	2				
D	0	1	2	3			
E	0	1	2	3	4		
F	0	1	2	3	4	5	
G		1	2	3	4	5	
H			2	3	4	5	
I				3	4	5	
J					4	5	
K						5	

Glorot et al. [50] found that while the new feature representations generated by the SDAs did not reduce $\mathcal{A}$-distance between domains, they did 'disentangle' information relating to domain from information relating to class. In the new representation, they found that the features that were most informative for determining class tended to be uninformative for determining domain and vice versa.

The hypothesis is that the features that the classifier finds most useful for the classification task will contain little or no domain information and therefore the performance of a classifier trained on the new representation will be less sensitive to domain. Fur-

thermore that this disentangling might be greatest in the higher mSDA layers, hence the inclusion of representations G to K where the lower layers are gradually dropped from the representation.

6.2.6 Classifier parameters

An SVM classifier with linear kernel was used, the same approach as Chen et al. [29]. The linear-SVM classifier regularisation parameter C was optimised using cross-validation for each iteration. This was felt to be necessary because of the the varying size of the training set and the varying number of features in the different representations.

However, as pointed out in Section 6.2.5, Glorot et al. [50] showed that the SDA representation tended to disentangle information important to classification from information important to domain. By setting the amount of regularisation to *only* maximise classification accuracy, the classifier may be using features that still carry *some* domain information and so this may reduce the domain adaptation impact of the SDA representation. This is discussed further at the end of this chapter.

6.2.7 Dataset samples and methods

As in Chapter 5 the training and (biased) test sets were drawn from the datasets. 320 such 'dataset-sample' iterations were carried out for each dataset giving 1,280 dataset-samples for UCI_dev, 960 for UCI_test and 320 for TAF. For each dataset-sample, quantification was carried out using each of 60 mSDA parameter combinations plus the baseline. The biassing method and parameters were the same as in Section 5.2.

6.3 Results

On the UCI_dev datasets, 29 of the 60 method-parameter combinations had a mean rank that was better than the baseline and where the null-hypothesis of no difference to the baseline could be rejected using the Friedman test with Bonferroni corrections at an $\alpha < 0.05$. The results of the best performing mSDA methods on the UCI_dev datasets are given in Table 6.2.

Table 6.2: Best mSDA methods on UCI_dev datasets against mean rank, RMSE and MAE

Method	First Layer	Last Layer	Noise	T-S Ratio	Mean Rank	RMSE	MAE	ΔMAE Abs.	ΔMAE Rel.
baseline					31.8	0.143	0.094		
mSDA	0	5	0.2		**26.7**	0.136	0.087	0.007	7.4%
mSDA	0	5	0.5		27.8	**0.135**	0.088	0.006	6.4%
mSDA	0	4	0.2	0.8	27.0	0.135	**0.087**	0.007	7.4%

Table 6.3 shows the performance on the UCI_test datasets of the methods identified as best on the UCI_dev datasets in Table 6.2.

Table 6.3: Best mSDA methods from UCI_dev measured on the UCI_test datasets

Method	Best On	First Layer	Last Layer	Noise	T-S Ratio	MAE	ΔMAE Abs.	ΔMAE Rel.	Stat Sig
baseline						0.151			
mSDA	Rank	0	5	0.2		0.149	0.002	1.3%	0.51
mSDA	MAE	0	4	0.2	0.8	0.146	0.005	3.3%	**0.02**
mSDA	RMSE	0	5	0.5		0.151	0.000	0.0%	0.90

Statistical significance is measured using a Wilcoxon signed-rank test [31]. While one of the best methods does give a statistically significant improvement in MAE against the baseline (with a p-value of 0.02), the improvement in MAE is modest and arguably, as with the Bonferroni corrections, the test of significance should take into account that *three* methods have been tested.

The impact of the mSDA parameters on quantification performance with the UCI_dev datasets is analysed below.

6.3.1 Layers

If generating a new feature representation with the mSDA architecture reduces the distance between domains then it is reasonable to think that that distance reduces at the higher

mSDA layers and therefore that the performance of a classifier trained on those higher layers would be more domain independent.

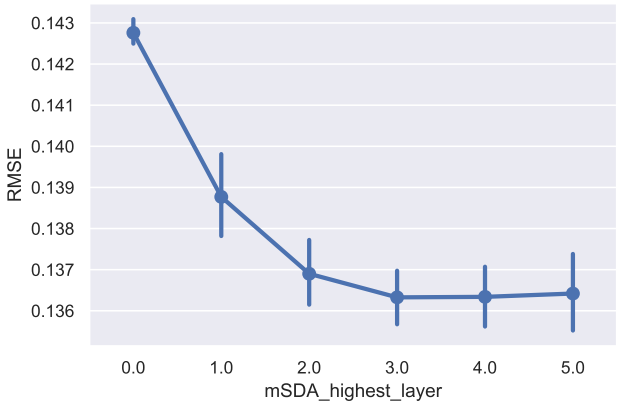

Figure 6.2: RMSE of results from UCI_dev datasets. mSDA layers from 0 to the layer indicated. 95% confidence intervals shown.

Figure 6.2 indicates that the addition of additional mSDA layers *does* reduce quantification error as measured by RMSE, but that there is no additional benefit obtained by adding layers above layer 3.

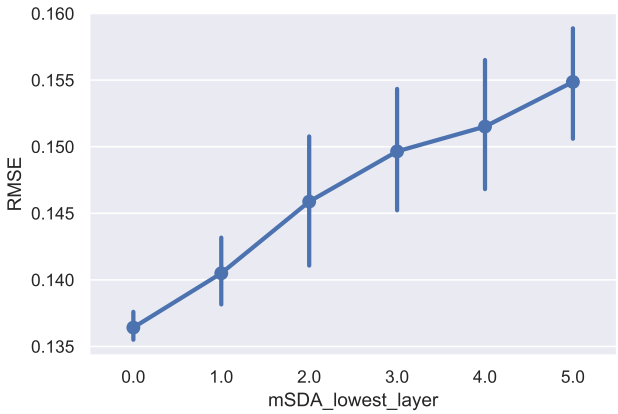

Figure 6.3: RMSE of results from UCI_dev datasets. mSDA layers from the layer indicated up to and including layer 5. 95% confidence intervals shown.

However, contrary to our hypothesis, Figure 6.3 shows that removing the lower mSDA layers (starting with layer 0 the original features) *reduces*, rather than improves performance.

6.3.2 Noise

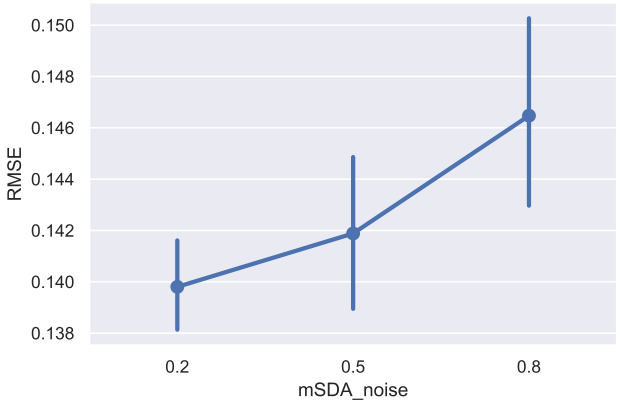

Figure 6.4: RMSE of results from UCI_dev datasets vs. level of noise. 95% confidence intervals shown.

Figure 6.4 indicates that performance is better with lower levels of noise, although this should be considered with caution given the overlapping confidence intervals. Further work would be needed to establish the optimum level of noise.

6.3.3 Oversampling from the Target domain

In Section 6.2 the hypothesis was put forward that adaptation between the Source and Target domains would work better when similar amounts of unlabelled data were used from both domains for training the mSDA.

Figure 6.5 indicates that adding oversampled data from the Target domain has no statistically significant effect.

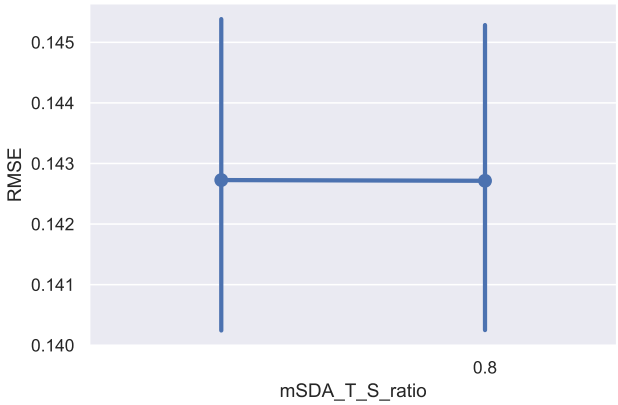

Figure 6.5: RMSE of results from UCI_dev datasets. Impact of the balance of mSDA training data between Source and Target domains. 95% confidence intervals shown.

The RMSE in cases where additional data was sampled from the Target domain to make this Target domain 0.8 times the size of the Source domain was not different from the RMSE observed when no additional data was added.

Given the negative impact of increasing the number of data instances on mSDA speed shown in Figure 6.1, it is useful to know that adding additional Target domain instances through over-sampling does *not* appear to give any performance improvement.

6.3.4 Training and test set size

The expectation was that larger datasets would give better results. However, Table 6.4 indicates that there is *no strong relationship* between the size of the training and test sets and performance.

Table 6.4: RMSE from best method on UCI_dev datasets vs. baseline for varying tr and te set sizes

te_size	tr_size	RMSE best	RMSE baseline	ΔRMSE Abs.	ΔRMSE Rel.
800	1200	0.127	0.135	0.008	5.9%
800	1600	0.136	0.144	0.008	5.6%
400	1200	0.154	0.164	0.010	6.1%
400	1600	0.124	0.128	0.004	3.1%

6.3.5 Recall

As discussed earlier in Section 6.2, the hypothesis at the core of this chapter is that a classifier trained on the SDA-transformed Source data would have less of a recall difference between the Source and Target domains than a classifier trained on the original features. With that classifier used in a *classify and adjust* quantifier, the smaller recall difference should translate into better quantification accuracy in the Target domain.

For analysis a single metric, 'recall_delta', has been computed for the difference in recall between the Target and Source domains:

$$\text{recall\_delta} = \sqrt{(r_{0T} - r_{0S})^2 + (r_{1T} - r_{1S})^2}, \tag{6.1}$$

where r_{cD} is the recall for class c in domain D.

The recall values for the Source domain were computed from the training set with cross-validation. The recall values for the Target domain were computed from the test set.

Figure 6.6 shows the observed relationship between absolute quantification error and recall_delta.

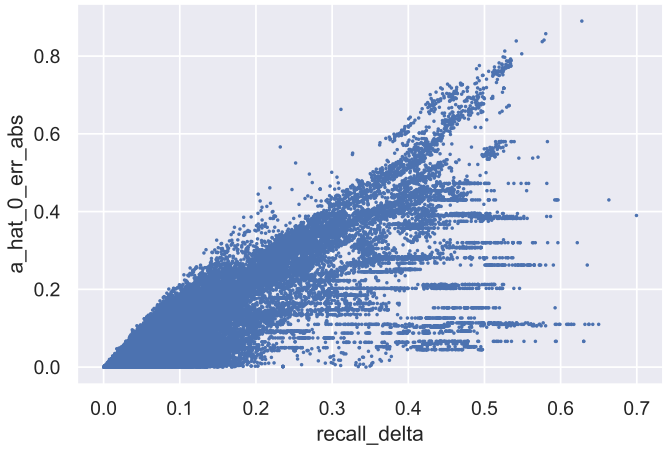

Figure 6.6: Absolute quantification error vs. recall_delta. mSDA results. UCI_dev datasets

Figure 6.6 indicates that higher quantification accuracy can be obtained when the difference in classifier recall between the domains is small.

Figures 6.7 and 6.8 show how recall_delta changes with the construction of the feature

representation from the mSDA layers.

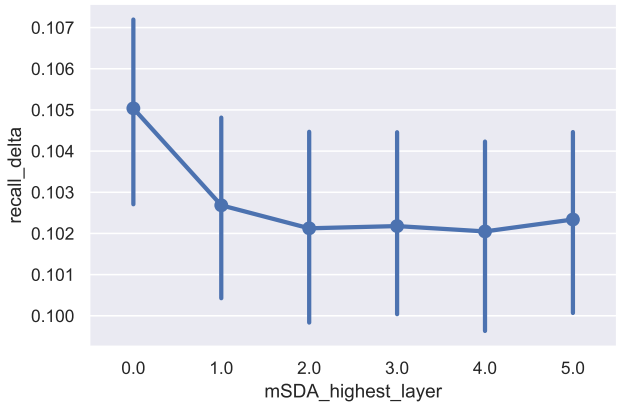

Figure 6.7: Recall_delta vs. mSDA highest layer for mSDA results with UCI_dev datasets. Lowest layer = layer 0

It shows that while adding additional layers initially reduces recall_delta, there appears to be no reduction in recall_delta after adding a second mSDA layer. The mean value for recall_delta does not drop below 0.102.

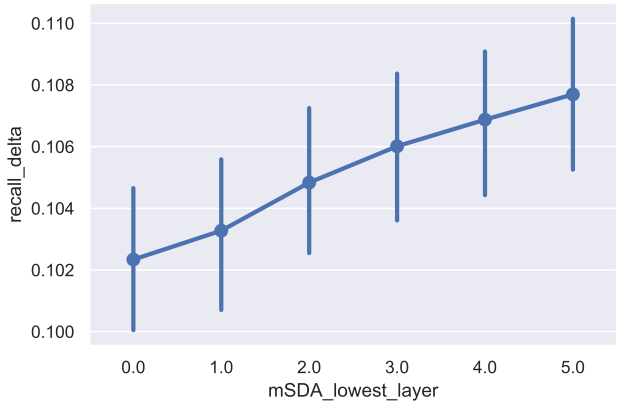

Figure 6.8: Recall_delta vs. mSDA lowest layer for mSDA results with UCI_dev datasets. Highest layer = layer 5

It was hypothesised that classifier recall may be more domain independent in the higher mSDA layers, however Figure 6.8 shows that removing lower layers from the feature representation *increases* the recall difference between the Target and Source domains.

6.3.6 By level of bias and by dataset

From Table 6.3, the best mSDA method has layers 0 to 5 and a noise setting of 0.2. Figures 6.9 and 6.10 show the quantification performance of this method with the UCI_dev and UCI_test datasets respectively.

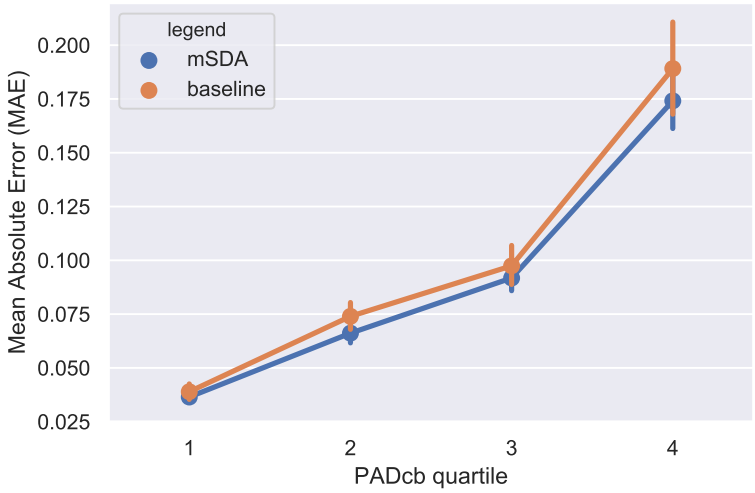

Figure 6.9: MAE of best mSDA method and baseline method vs. PADcb quartile. UCI_dev datasets. 95% confidence intervals

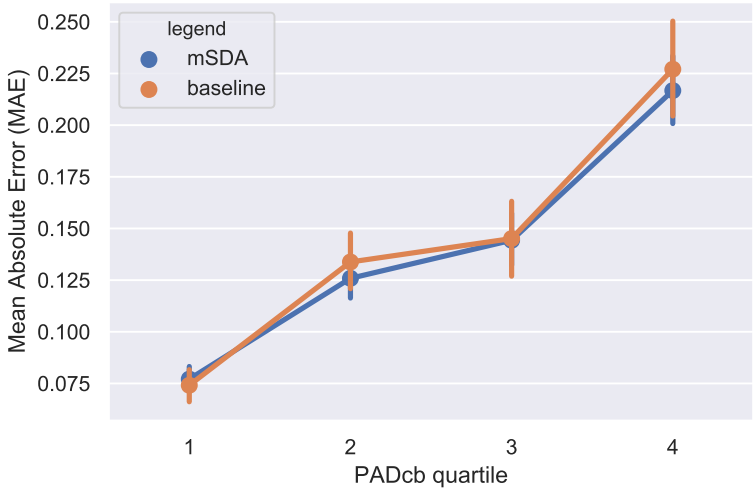

Figure 6.10: MAE of best mSDA method and baseline method vs. PADcb quartile. UCI_test datasets. 95% confidence intervals

As would be expected, the performance vs. baseline is in general better at higher levels of

dataset bias. The level of improvement from the best mSDA method is not as large as was seen earlier in Figures 5.9 and 5.10 with instance weighting using Kernel Mean Matching.

Figures 6.11 and 6.12 show the difference in quantification performance for the best mSDA method against the baseline vs. dataset bias for the individual datasets in UCI_dev and UCI_test respectively. Values above zero indicate that the method is worse than the baseline, values below zero indicate that the method is better than the baseline.

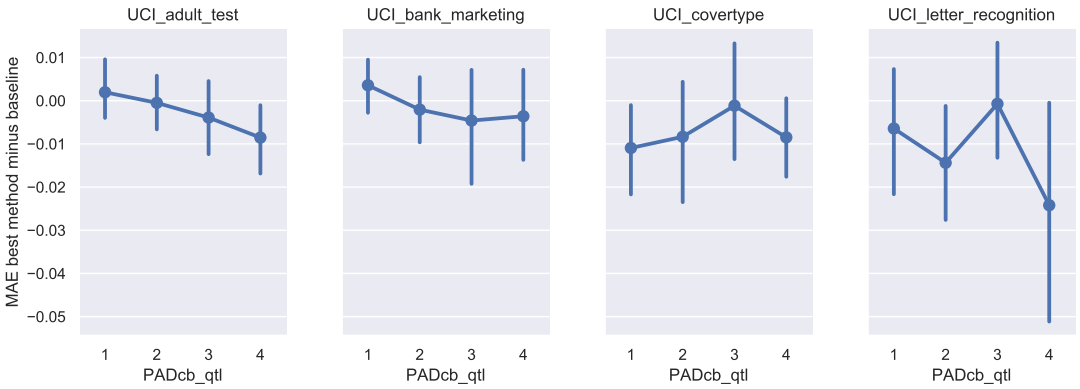

Figure 6.11: MAE of 'best' method minus MAE baseline by PADcb quartile for UCI_dev datasets

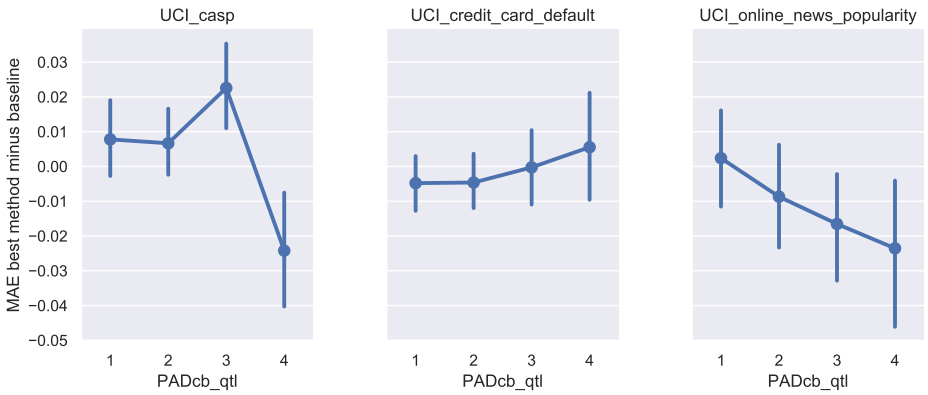

Figure 6.12: MAE of 'best' method minus MAE baseline by PADcb quartile for UCI_test datasets

As in the previous chapter, the best method has been judged on the basis of average performance over the distribution of biased datasets that was generated in the experiments by the biassing method. In deployment the best method will depend on the expected level of bias.

6.3.7 TAF

Table 6.5 shows the results of applying the best methods as determined on the UCI_dev datasets (Table 6.2) to the TAF dataset.

Table 6.5: Best mSDA methods from UCI_dev measured on the TAF dataset

Method	Best on	First Layer	Last Layer	Noise	T-S Ratio	MAE	ΔMAE Abs.	ΔMAE Rel.	Stat Sig
baseline						0.085			
mSDA	Rank	0	5	0.2		0.080	0.005	5.9%	0.22
mSDA	MAE	0	4	0.2	0.8	0.082	0.003	3.5%	0.52
mSDA	RMSE	0	5	0.5		0.084	0.001	1.2%	0.97

While these methods gave a quantification accuracy that was better than the baseline accuracy, the null-hypothesis of no-difference could not be reliably rejected. The Friedman-Bonferroni test was run on the TAF dataset. Again, a number of methods performed better than the baseline none were statistically significantly different to the baseline method. The results are shown in Table 6.6.

Table 6.6: Best mSDA methods on TAF dataset against mean rank, RMSE and MAE

Method	First Layer	Last Layer	Noise	T-S Ratio	Mean Rank	RMSE	MAE	ΔMAE Abs.	ΔMAE Rel.
baseline					29.9	0.113	0.085		
mSDA	0	3	0.5	0.8	**26.9**	0.102	0.078	0.007	8.2%
mSDA	0	4	0.5	0.8	27.5	**0.100**	**0.078**	0.007	8.2%

Figure 6.13 shows that, as with the UCI_dev and UCI_test datasets, the mSDA method gives a larger improvement over the baseline at higher levels of dataset bias. For comparison purposes the PADcb quartile values used in Figure 6.13 are those derived from the UCI_dev datasets *not* the quartiles from the TAF dataset itself.

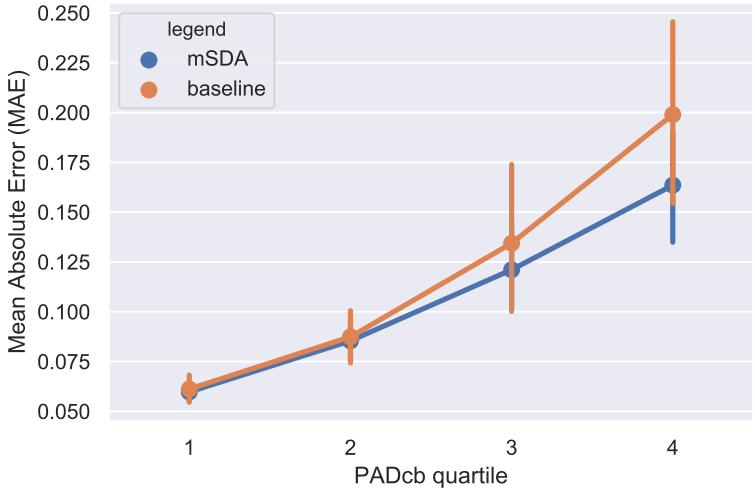

Figure 6.13: MAE of best mSDA method and baseline method vs. PADcb quartile. TAF dataset. 95% confidence intervals

6.4 Conclusions

Table 6.7 gives a summary of the results.

Table 6.7: Summary of Chapter 6 results

		Method	Set On	Measured On	ΔMAE Absolute	ΔMAE Relative	Stat Sig[2]
Tab.	6.2	mSDA	UCI_dev	UCI_dev	0.7%	7.4%	yes
Tab.	6.3	mSDA	UCI_dev	UCI_test	0.5%	3.3%	yes
Tab.	6.5	mSDA	UCI_dev	TAF	0.5%	5.9%	no
Tab.	6.6	mSDA	TAF	TAF	0.7%	8.2%	no

The best parameter settings found using UCI_dev gave an MAE that was 0.5% points lower than the baseline method on the held-out UCI_test datasets. This was a relative improvement in MAE of 3.3%.

It is possible that some further improvement could be obtained from the mSDA *feature*

[2]Was this finding statistically significant

representation methods. In particular, looking at feature selection and regularisation to actively minimise the influence of domain information on the classifier.

It would be interesting to look further at the impact of noise level on the results and also to explore whether adding genuine (rather than over-sampled) unlabelled instances from the Target domain has an impact. If additional instances are unavailable then potentially the SMOTE method [27] could be used to generate synthetic datapoints for the Target domain.

While the mSDA method [29] is much quicker to train than the original neural-network based SDA method [50], as shown in Figure 6.1 it still takes time to train and it does not scale well to larger datasets. Unless other unlabelled data is already available from the Target domain this method has the same run-time issue as the instance weighting based method i.e. that the classifiers for quantification cannot be trained until the feature representation has been trained and this cannot happen until the test set data is presented.

Chapter 7

Conclusions and further work

The question that this work has aimed to address is:

How do you accurately estimate the class proportions in a dataset when the class-conditional feature distribution is different to that of the dataset that is available for training?

Or more mathematically, how do we accurately estimate $P_T(y)$ when $P_S(y)$ is known but $P_S(x|y) \neq P_T(x|y)$?

All of the approaches that were taken in this thesis followed the same basic structure: reduce the difference in class-conditional feature distributions so that a standard *classify and adjust* quantification method can work effectively:

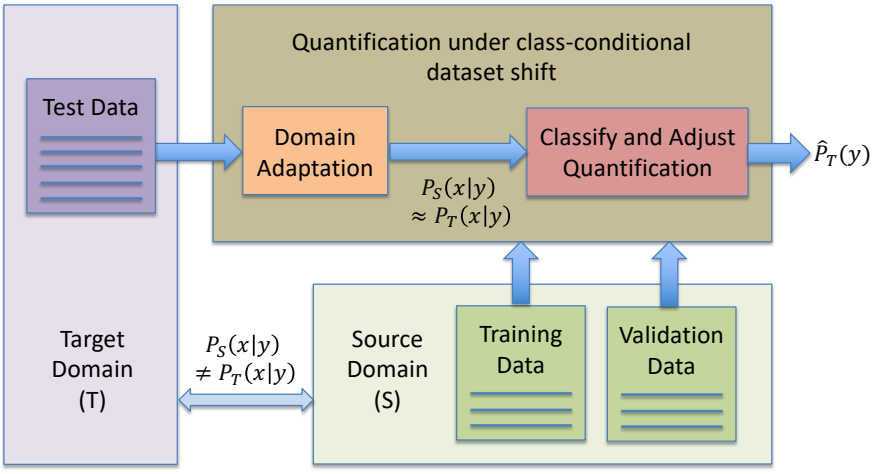

Figure 7.1: Common approach to quantification under class-conditional dataset shift

Standard *classify and adjust* quantification methods were used while the focus was on the *domain adaptation* step: how to bring the class-conditional feature distributions from the two domains as close together as possible. Three quite different approaches were taken to the *domain adaptation* problem: *Explicit Sub-Domains* (ESD) in Chapters 3 and 4, *Instance Weighting* (IW) in Chapter 5 and *Feature Representations* (FW) in Chapter 6.

A full summary of the results of the various experiments is given in Table 7.1.

Table 7.1: Summary of results

		Method	Set on	Measured on	ΔMAE Abs.	ΔMAE Rel.	Stat Sig[1]
Tab. 4.9	ESD	tsd single	UCI_dev	UCI_test	0.42%	3.7%	-
Tab. 4.9	ESD	tsd multiple	UCI_dev	UCI_test	0.51%	4.5%	-
Tab. 4.19	ESD	tsd single	UCI_dev	TAF	0.03%	0.3%	-
Tab. 4.19	ESD	tsd multiple	UCI_dev	TAF	0.07%	0.7%	-
Tab. 5.9	IW	KMMut0.5	UCI_dev	UCI_test	1.3%	10.7%	yes
Tab. 5.10	IW	uLSIFww	UCI_dev	UCI_dev	0.5%	5.0%	yes
Tab. 5.11	IW	SSBCut0.9	UCI_dev	UCI_dev	0.2%	1.7%	no
Tab. 6.2	FR	mSDA	UCI_dev	UCI_dev	0.7%	7.4%	yes
Tab. 6.3	FR	mSDA	UCI_dev	UCI_test	0.5%	3.3%	yes
Tab. 6.5	FR	mSDA	UCI_dev	TAF	0.5%	5.9%	no
Tab. 6.6	FR	mSDA	TAF	TAF	0.7%	8.2%	no

All the results in Table 7.1 are averages over the distribution of biased test data generated by the experiments.

Caution should be exercised when comparing results from *explicit sub-domains* (ESD) with the results from *instance weighting* (IW) and *feature representations* (FR). Firstly, with *explicit sub-domains* the feature that was used for biassing the datasets was *also* the feature that defined the sub-domains. While in some situations the sub-domain where bias has occurred is obvious (e.g. older people in Scotland) in other cases it may be less so. Had the sub-domains been specified *without* knowledge of the feature used for biassing then it is likely that the results would not be as good. Secondly, the method of biasing the

[1] Result is statistically significant

test sets was different. With *explicit sub-domains* in Chapters 3 and 4, the biased datasets were constructed by sampling to a given sub-domain proportion. In Chapters 5 on *instance weighting* and 6 on *feature representations*, biassing was done with the method used by Gretton et al. [58] with parameters that remained constant on all tests as described in Section 5.2.

7.1 Datasets and bias

Biassing of the test sets was carried out with a consistent method and parameters but generated different distributions of test sets by bias for each dataset as shown in Figures 5.1. The *input* in terms of method and parameters was effectively constant but the *outcome* varied by dataset. It would be quite feasible to sample the test sets to give constant *outcome* distributions of bias for each dataset but then the question arises as to what distribution is 'correct'? I am not aware of any work that has been done to establish the degree to which the artificial bias used in the test datasets is representative of the bias that would be encountered in real applications.

This would also be an interesting area of further study. It would be possible to analyse the sets of users generated by projects that have been carried out at the University of Sussex using the Method 52 tool. The observed bias between these groups and the populations from which they are drawn could be compared to the levels of bias generated by the method used in this thesis.

Twitter had provided the motivation the the initial experiment in Chapter 3 so it made sense to return to Twitter in the later chapters. In Chapter 4, results on the TAF dataset were an order of magnitude *worse* than those obtained on the UCI_test datasets, but in Chapter 6 the results on the TAF dataset were *comparable* to the results with the UCI_test datasets. This may be due to the method of biassing. In Chapter 6 the Gretton et al. [58] method was used, while in Chapter 4 the TAF dataset was biased using the annotated sub-domains shown in Figure 4.12. While some of those categories have large difference in recall, e.g. 'Asia' in location, many do not. It may simply be that the test sets in Chapter 4 were not particularly biased. This hypothesis is supported by the finding that the set thresholds achieved a performance that was 55% of the optimal performance for the dataset i.e. there was not much room for improvement above the baseline, which would be consistent with a low level of bias in the test sets.

7.2 Explicit subdomains

With *explicit sub-domains* the assumption is made that the data can be broken down into smaller groups ('sub-domains') in which the conditional feature distributions *do not* vary i.e.:

$$P_S(x|y, sd) \approx P_T(x|y, sd) \tag{7.1}$$

The difference in class-conditional feature distributions between the two domains is then explained by different proportions of sub-domains. Quantification is carried out at the sub-domain level and the results then aggregated up to class level as a final step.

It appeared that a closed-form solution *might* be possible for the expected error when doing quantification by matrix-inversion with explicit sub-domains. It was important to explore this, at the very least to check that the solution was not trivial. As it turned out, the solution was not trivial. In this thesis, a closed-form expression could only be found for the case where the validation set was assumed to be large. An obvious piece of further work would be to obtain a general solution without that simplifying assumption, however it is still not clear whether the more complex mathematics that would be required would deliver a *useful* closed-form expression. To be useful the closed form expression should indicate whether the sd-method or nsd-method will give the lowest quantification error based on parameters that would be known in advance such as the dataset sizes and the classifier recall values in the Source domain.

The fact that the solution was non-trivial justified the use of simulation in Section 3.8. Simulation showed the impact of separate parameters on the expected quantification error. These insights were then used to formulate the thresholded sub-domain method in Chapter 4. The hypothesis at the start of this chapter was that while it appeared that the size of the validation set could be a good threshold for when to apply explicit sub-domains and when not to apply them, it may be be difficult to define a value for *sufficiently large* that works for all potential domains that may be encountered. It was felt that a more principled threshold based on the significance of the recall difference between sub-domains might be more reliable across different domains.

However, the threshold at which using explicit sub-domains (sd-method) gave better quantification accuracy proved to be quite consistent across a number of datasets as shown in Table 7.2.

Table 7.2: Size of validation set above which the sd-method gave better quantification accuracy than the nsd-method

Section	Dataset	n_v Threshold	Note
3.8.3	Simulated	≈ 1000	
4.1.3.1	Simulated	≈ 1000	
4.2.6.1	UCI_dev datasets	≈ 1000	All[2]
4.3.1	UCI_dev dataset	≈ 316	Average[3]
4.4.4	TAF	≈ 1000	Optimal for TAF

It would be interesting to explore this relationship further and see if a principled link can be established between quantification accuracy and validation set size. While validation set size was the best single criteria to use, combining it with other criteria did lead to be better quantification accuracy.

In this thesis, the sub-domains were *explicit*. Several published works ([5][62][64]) explored domain adaptation through *latent* sub-domains as a way of improving *classification* and these were reviewed in Section 2.3.1.2. Typically in these works, the class-proportion in the test set was estimated as an intermediate step towards improving classification. While some of the published results of these methods are not completely compelling it would still be worthwhile exploring if they can give low *quantification* error under class-conditional dataset shift. The method in Hofer [62] appears to perform well but was not implemented for the reasons set out in Section 2.3.1.2. However, benchmarking this method against the methods in this thesis would be an interesting piece of further work.

As stated earlier, the explicit sub-domains methods used relied on the sub-domain being the same as the label used for biassing the dataset. It would be quite possible to devise a method which identified the best feature for use as the sub-domain. The best feature would be one that combined a difference in distribution between the validation dataset and the test dataset and a difference in main-class recall between the different values of the feature in the validation data.

[2] Above this threshold value the sd-method gave better accuracy on all four datasets

[3] Above this threshold value the sd-method gave better accuracy on average across the four datasets

Instead of individual features, a better way to identify sub-domains might be to look at clusters of 'similar' users. With a user typically following only a few hundred other accounts of the many millions available, the information about the user from *following* an account is not the simple inverse of *not following* that account. There are a range of possible techniques that could be used for finding clusters in this type of *categorical* data including Rock [59], Limbo [9], Chameleon [75], Clicks [123] or Coolcat [10]. Alternatively, network concepts such as modularity [89] might lead to the identification of meaningful sub-domains.

7.3 Importance weighting

Weights w were computed for the instances in the training data (from the Source domain) and used to bring the class-conditional feature distributions from the Target and Source domains close together:

$$P_S(x|y, w) \approx P_T(x|y) \tag{7.2}$$

The KMM, uLSIF and SSBC instance weighting methods are *not* inherently class balanced. The methods compute the instance weights that will minimise the difference in the joint distribution $P(x, y)$ between the Source and Target domains under the covariate shift assumption that $P(y|x)$ is constant. With Bayes rule we can express the joint distribution as $P(x, y) = P(x|y)P(y)$. The results of Section 5.6 show a correlation between the weights and the marginal class distributions. For quantification we ideally want the weights to align the conditional feature distributions, $P_S(x|y) = P_T(x|y)$, but *not* to align the marginal class distributions $P_S(y)$ and $P_T(y)$. Table 5.5 set out the four approaches were adopted to try and achieve this. *All* of the methods that had a quantification accuracy that was significantly better than the baseline were class-balanced using one of the four approaches.

Applying the computed weights directly to the instances in the training set should minimise the difference between the Source and Target distributions. However, the method that actually gave the best quantification accuracy used KMM-computed weights in a thresholded way to select the 50% of the validation set that was most similar to the test data from the Target domain. *All* of the training set was used, unweighted, to train the classifier. The best parameter settings gave a mean absolute quantification error that was 10.7% lower than the baseline method. Initial expectations had been low for the KMM

method after the original authors in Gretton et al. [58] had shown poor results. However, KMM-based methods out-performed methods that used weights computed by uLSIF and SSBC. The relatively poor performance of the SSBC method from Section 5.8.2 was not that surprising. The method pre-dates the KMM and uLSIF methods and where this method has been evaluated by other authors it has generally not been the best performer. However, the uLSIF method is more modern than the KMM method and in Sugiyama and Kawanabe [104] the authors had said that in comparison with other methods including KMM that uLSIF is a preferable method for importance estimation.

One of the four approaches to class-balancing was the *weakly supervised* approach. In a range of experiments it was not possible to get this approach to achieve a quantification accuracy that was better than the baseline. Despite this, I think there may be some merit in pursuing the *weakly supervised* methods further. With training data consisting of both labelled data from the Source domain *and* some (weakly) labelled data from the Target domain, direct distribution matching separately by class would be possible.

Better than *weakly supervised*, if some *actual* labelled data was available in the Target domain then *semi-supervised* approaches could be applied. With Twitter users this could well be possible. If we want to quantify by age group then (as described in Section 3.4.1) it is quite likely that *some* users in the test set will have usernames that end in a number that is a plausible year of birth. Similarly if we want to quantify by gender then *some* of the instances in the Target domain could conceivably be gender labelled with a method that uses first names. With *some* labelled data in the Target domain it would be possible to apply instance weighting class-conditionally.

Both the KMM and uLSIF importance weighting methods used Gaussian kernels. The L_2-norm (Euclidean distance) is fundamental to the Gaussian kernel, but Aggarwal et al. [3] casts doubt on the L_2-norm as a distance measure with high-dimensional data. Given the high dimensionality of the motivating problem of Twitter users and the accounts that they follow, it would be interesting to see whether alternative measures of distance would yield better results. Euclidean distance in a sparse categorical feature space does not make much sense. It would be interesting to explore whether categorical clustering methods such as those in Section 7.2 would give insights on finding training data that was close to the test data.

7.4 Feature representation

The *feature representations* approach was to generate a new representation x', and train a classifier on this representation with hypothesis h such that:

$$P_S(x'|y,h) \approx P_T(x'|y,h) \tag{7.3}$$

While the results are not strictly comparable with the results from the *explicit sub-domains* experiments in Chapters 3 and 4, they are comparable with the *instance weighting* results from Chapter 5. The mSDA feature representation approach is not as strong a that of the best KMM based method from Chapter 5. The best paramater settings gave a mean absolute quantification error that was 3.3% lower than the baseline method.

One area where further work would quite conceivably improve performance is in the area of regularisation and feature selection. As stated in Sections 6.2.5 and 6.2.6, Glorot et al. [50] found that the SDA approach did not remove information on domain but rather 'disentangled' it from information on class. Features that were most informative for class were not very informative for domain and vice versa. In this work, the classifier had access to *all* features in the given representation and regularisation was set by cross validation to *only* maximise classification accuracy.

It may be better to conduct feature selection and/or regularisation to give good classification accuracy *but at the same time* to minimise the information relating to domain in the weighted features. The method used by Donini et al. [34] could be tried. There are also potential overlaps with the adversarial methods discussed in Section 2.3.3.4 and with multi-task learning [22].

The only feature representation approach used in this thesis was Stacked De-noising Autoencoder method from Glorot et al. [50] and Chen et al. [29]. As discussed in Section 2.3.3 a range of other methods of generating feature representations for domain adaptation are available and could be explored. The *Quantification Trees* approach from Milli et al. [86] was reviewed in Section 2.1.4. I speculated that a tree based approach to quantify a set of instances might be robust to shifts in class-conditional feature distributions. This could be a dead-end but given the success that boosted tree-based approaches for classification (e.g. CatBoost, XGBoost) it could also be an interesting line of research.

Adversarial methods using neural networks (e.g. [47]) are currently regarded as state of the art for domain adaptation. They are discussed in Section 2.3.3.4. If they are able to

build a feature representation from which it is difficult to detect domain then a *classify and adjust* quantifier trained on that representation should be domain independent. However, the results from the ITTBR method in 5.9 showed that minimising the Proxy $\mathcal{A}$-distance between domains *does not necessarily* result in classifier performance being independent of domain.

There is common ground between the work in this thesis and in ongoing work on fairness at the University of Sussex. If domains are thought of as representing people from various protected categories (e.g. age, gender, race), an aspect of 'fairness' is that classification decisions are domain independent.

7.5 Direct quantification with biased training sets

Direct quantification methods were discussed in Section 2.1.4. In these methods a quantifier is trained on training sets of known class distributions, and the loss function for training is linked to the quantification error on a set as opposed to (or as well as [12]) the classification error on individual instances. From the published work it does not appear that anyone had tried to train direct quantification methods to cope with class-conditional feature distribution changes using biased training sets.

One potential approach would be to train a multi-layer neural network to estimate class proportions in sets of instances where each training set would be artificially biased. Biassing methods could be the same as those used to generate biased test sets in Section 5.2. The neural network would require some function to generate a single feature representation from a labelled training set. *Instance weighting* and *feature representation* methods work by performing some function with data from *both* the Source domain and the particular Target domain. In this method a quantifier learns from artificially biased training sets from the Source domain. This raises interesting questions. To what degree can such a quantifier generalise from the artificially biased sets on which it was trained, to an unseen, 'naturally' [38] biased set from the Target domain? Is it possible (and useful) to train the quantifier both on sets of artificially biased labelled training data *and* on the unlabelled data from the Target domain? How could you be certain that that the method was generalising and not simply 'learning' the specific method of creating biased datasets, especially if similar methods are used for biassing both the training and the test sets? Clearly there is a strong link to the discussion on bias in reality in Section 7.1.

7.6 Implementation

Both the KMM based, thresholded method and the mSDA method would be straightforward to implement in an operational system. There are a number of considerations. Both approaches learn using the unlabelled test data so there will inevitably be some time delay between the presentation of the test data and the computation of the estimated class proportions. Both methods are known to have issues with scaling and dimensionality. For the mSDA method, an approach to dealing with this is in the original paper by Chen et al. [29]. Depending on the application area, some work may be required to implement an approach that is sufficiently fast.

7.7 Last words...

'Unsupervised domain adaptation is an inherently hard problem' [55]. Many seemingly promising methods failed to improve on the performance of a simple *classify and adjust* baseline and were rejected. However, all three of the approaches taken in this thesis, *explicit subdomains*, *instance weighting* and *feature representations* did eventually deliver methods that improved on the baseline quantification performance. Of the three, a method using Kernel Mean Matching to weight instances in the validation data showed the best performance.

Chapter 8

Bibliography

[1] The distribution of a ratio of correlated normals. `https://davegiles.blogspot.com/2015/08/the-distribution-of-ratio-of-correlated.html`. [Online; accessed 25-September-2018].

[2] Friedman's two-way analysis of variance by ranks — analysis of k-within-group data with a quantitative response variable. `http://psych.unl.edu/psycrs/handcomp/hcfried.PDF`. [Online; accessed 30-August-2018].

[3] Charu C Aggarwal, Alexander Hinneburg, and Daniel A Keim. On the surprising behavior of distance metrics in high dimensional space. In *International Conference on Database Theory*, pages 420–434. Springer, 2001.

[4] Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, and Mario Marchand. Domain-adversarial neural networks. *arXiv preprint arXiv:1412.4446*, 2014.

[5] Rocío Alaiz-Rodríguez, Alicia Guerrero-Curieses, and Jesús Cid-Sueiro. Class and subclass probability re-estimation to adapt a classifier in the presence of concept drift. *Neurocomputing*, 74(16):2614–2623, 2011.

[6] Giambattista Amati, Simone Angelini, Marco Bianchi, Luca Costantini, and Giuseppe Marcone. A cumulative approach to quantification for sentiment analysis. *arXiv preprint arXiv:1610.01366*, 2016.

[7] Martin Andersen, Joachim Dahl, and Lieven Vandenberghe. Cvxopt. `https://cvxopt.org/`. [Online; accessed 31-August-2018].

[8] Rie Kubota Ando and Tong Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6 (Nov):1817–1853, 2005.

[9] Periklis Andritsos, Panayiotis Tsaparas, Renée J Miller, and Kenneth C Sevcik. Limbo: Scalable clustering of categorical data. In *EDBT*, pages 123–146. Springer, 2004.

[10] Daniel Barbará, Yi Li, and Julia Couto. Coolcat: an entropy-based algorithm for categorical clustering. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 582–589. ACM, 2002.

[11] Jose Barranquero, Pablo González, Jorge Díez, and Juan José Del Coz. On the study of nearest neighbor algorithms for prevalence estimation in binary problems. *Pattern Recognition*, 46(2):472–482, 2013.

[12] Jose Barranquero, Jorge Díez, and Juan José del Coz. Quantification-oriented learning based on reliable classifiers. *Pattern Recognition*, 48(2):591–604, 2015.

[13] Antonio Bella, Cesar Ferri, José Hernández-Orallo, and Maria Jose Ramirez-Quintana. Quantification via probability estimators. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 737–742. IEEE, 2010.

[14] Shai Ben-David, John Blitzer, Koby Crammer, Fernando Pereira, et al. Analysis of representations for domain adaptation. *Advances in neural information processing systems*, 19:137, 2007.

[15] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[16] Steffen Bickel, Michael Brückner, and Tobias Scheffer. Discriminative learning for differing training and test distributions. In *Proceedings of the 24th international conference on Machine learning*, pages 81–88. ACM, 2007.

[17] Christophe Biernacki, Farid Beninel, and Vincent Bretagnolle. A generalized discriminant rule when training population and test population differ on their descriptive parameters. *Biometrics*, 58(2):387–397, 2002.

[18] John Blitzer, Ryan McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 conference on empirical*

methods in natural language processing, pages 120–128. Association for Computational Linguistics, 2006.

[19] John Blitzer, Mark Dredze, Fernando Pereira, et al. Biographies, bollywood, boomboxes and blenders: Domain adaptation for sentiment classification. In *ACL*, volume 7, pages 440–447, 2007.

[20] Karsten M Borgwardt, Arthur Gretton, Malte J Rasch, Hans-Peter Kriegel, Bernhard Schölkopf, and Alex J Smola. Integrating structured biological data by kernel maximum mean discrepancy. *Bioinformatics*, 22(14):e49–e57, 2006.

[21] AA Buck, JJ Gart, et al. Comparison of a screening test and a reference test in epidemiologic studies. ii. a probabilistic model for the comparison of diagnostic tests. *American Journal of Epidemiology*, 83(3):593–602, 1966.

[22] Rich Caruana. Multitask learning. In *Learning to learn*, pages 95–133. Springer, 1998.

[23] Tianfeng Chai and Roland R Draxler. Root mean square error (rmse) or mean absolute error (mae)?–arguments against avoiding rmse in the literature. *Geoscientific Model Development*, 7(3):1247–1250, 2014.

[24] Yee Seng Chan and Hwee Tou Ng. Word sense disambiguation with distribution estimation. In *IJCAI*, volume 5, pages 1010–5, 2005.

[25] Yee Seng Chan and Hwee Tou Ng. Estimating class priors in domain adaptation for word sense disambiguation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 89–96. Association for Computational Linguistics, 2006.

[26] Nitesh V Chawla. Data mining for imbalanced datasets: An overview. In *Data mining and knowledge discovery handbook*, pages 853–867. Springer, 2005.

[27] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[28] Minmin Chen, Yixin Chen, and Kilian Q Weinberger. Automatic feature decomposition for single view co-training. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 953–960, 2011.

[29] Minmin Chen, Zhixiang Xu, Kilian Weinberger, and Fei Sha. Marginalized denoising autoencoders for domain adaptation. *arXiv preprint arXiv:1206.4683*, 2012.

[30] David W Cowling, Ian A Gardner, and Wesley O Johnson. Comparison of methods for estimation of individual-level prevalence based on pooled samples. *Preventive veterinary medicine*, 39(3):211–225, 1999.

[31] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006.

[32] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

[33] Alan W Donald, Ian A Gardner, and Alvin D Wiggins. Cut-off points for aggreate herd testing in the presence of disease clustering and correlation of test errors. *Preventive Veterinary Medicine*, 19(3):167–187, 1994.

[34] Michele Donini, Luca Oneto, Shai Ben-David, John Shawe-Taylor, and Massimiliano Pontil. Empirical risk minimization under fairness constraints. *arXiv preprint arXiv:1802.08626*, 2018.

[35] Marthinus Christoffel Du Plessis and Masashi Sugiyama. Semi-supervised learning of class balance under class-prior change by distribution matching. *Neural Networks*, 50:110–119, 2014.

[36] Lixin Duan, Ivor W Tsang, Dong Xu, and Tat-Seng Chua. Domain adaptation from multiple sources via auxiliary classifiers. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 289–296. ACM, 2009.

[37] Andrea Esuli and Fabrizio Sebastiani. Optimizing text quantifiers for multivariate loss functions. *arXiv preprint arXiv:1502.05491*, 2015.

[38] Andrea Esuli, Fabrizio Sebastiani, and Ahmed ABBASI. Sentiment quantification. *IEEE intelligent systems*, 25(4):72–79, 2010.

[39] Tom Fawcett and Peter A Flach. A response to webb and ting's on the application of roc analysis to predict classification performance under varying class distributions. *Machine Learning*, 58(1):33–38, 2005.

[40] Basura Fernando, Amaury Habrard, Marc Sebban, and Tinne Tuytelaars. Unsupervised visual domain adaptation using subspace alignment. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2960–2967, 2013.

[41] Peter Flach. *Machine learning: the art and science of algorithms that make sense of data*. Cambridge University Press, 2012.

[42] George Forman. Counting positives accurately despite inaccurate classification. In *Machine Learning: ECML 2005*, pages 564–575. Springer, 2005.

[43] George Forman. Quantifying trends accurately despite classifier error and class imbalance. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 157–166. ACM, 2006.

[44] George Forman. Quantifying counts and costs via classification. *Data Mining and Knowledge Discovery*, 17(2):164–206, 2008.

[45] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):44, 2014.

[46] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. *arXiv preprint arXiv:1409.7495*, 2014.

[47] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 17(59):1–35, 2016.

[48] Wei Gao and Fabrizio Sebastiani. Tweet sentiment: From classification to quantification. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, pages 97–104. ACM, 2015.

[49] Wei Gao and Fabrizio Sebastiani. From classification to quantification in tweet sentiment analysis. *Social Network Analysis and Mining*, 6(1):1–22, 2016.

[50] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 513–520, 2011.

[51] Boqing Gong, Fei Sha, and Kristen Grauman. Overcoming dataset bias: An unsupervised domain adaptation approach. In *NIPS Workshop on Large Scale Visual Recognition and Retrieval*, volume 3, 2012.

[52] Pablo González, Alberto Castaño, Nitesh V Chawla, and Juan José Del Coz. A review on quantification learning. *ACM Computing Surveys (CSUR)*, 50(5):74, 2017.

[53] Pablo González, Jorge Díez, Nitesh Chawla, and Juan José del Coz. Why is quantification an interesting learning problem? *Progress in Artificial Intelligence*, 6(1): 53–58, 2017.

[54] VíCtor GonzáLez-Castro, RocíO Alaiz-RodríGuez, and Enrique Alegre. Class distribution estimation based on the hellinger distance. *Information Sciences*, 218: 146–164, 2013.

[55] Raghuraman Gopalan, Ruonan Li, and Rama Chellappa. Domain adaptation for object recognition: An unsupervised approach. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 999–1006. IEEE, 2011.

[56] M Greiner and IA Gardner. Application of diagnostic tests in veterinary epidemiologic studies. *Preventive veterinary medicine*, 45(1):43–59, 2000.

[57] Arthur Gretton, Karsten M Borgwardt, Malte Rasch, Bernhard Schölkopf, Alexander J Smola, et al. A kernel method for the two-sample-problem. *Advances in neural information processing systems*, 19:513, 2007.

[58] Arthur Gretton, Alex Smola, Jiayuan Huang, Marcel Schmittfull, Karsten Borgwardt, and Bernhard Schölkopf. Covariate shift by kernel mean matching. *Dataset shift in machine learning*, 3(4):5, 2009.

[59] Saikat Guha, Rajeev Rastogi, and Kyuseok Shim. Rock: A robust clustering algorithm for categorical attributes. In *Data Engineering, 1999. Proceedings., 15th International Conference on*, pages 512–521. IEEE, 1999.

[60] James J Heckman. Sample selection bias as a specification error (with an application to the estimation of labor supply functions), 1977.

[61] David V Hinkley. On the ratio of two correlated normal random variables. *Biometrika*, 56(3):635–639, 1969.

[62] Vera Hofer. Adapting a classification rule to local and global shift when only unlabelled data are available. *European Journal of Operational Research*, 243(1):177–189, 2015.

[63] Vera Hofer and Georg Krempl. Drift mining in data: A framework for addressing drift in classification. *Computational Statistics & Data Analysis*, 57(1):377–391, 2013.

[64] Judy Hoffman, Brian Kulis, Trevor Darrell, and Kate Saenko. Discovering latent domains for multisource domain adaptation. In *Computer Vision–ECCV 2012*, pages 702–715. Springer, 2012.

[65] Daniel J Hopkins and Gary King. A method of automated nonparametric content analysis for social science. *American Journal of Political Science*, 54(1):229–247, 2010.

[66] Jiayuan Huang, Arthur Gretton, Karsten M Borgwardt, Bernhard Schölkopf, and Alex J Smola. Correcting sample selection bias by unlabeled data. In *Advances in neural information processing systems*, pages 601–608, 2006.

[67] Chris Inskip. Inferring user demographics from social media. Undergraduate final year project, The University of Sussex, 2015.

[68] Arun Iyer, Saketha Nath, and Sunita Sarawagi. Maximum mean discrepancy for class ratio estimation: Convergence bounds and kernel selection. In *ICML*, pages 530–538, 2014.

[69] Nathalie Japkowicz. The class imbalance problem: Significance and strategies. In *Proc. of the Int'l Conf. on Artificial Intelligence*. Citeseer, 2000.

[70] Jing Jiang. A literature survey on domain adaptation of statistical classifiers. *URL: http://sifaka. cs. uiuc. edu/jiang4/domainadaptation/survey*, 2008.

[71] Jing Jiang and ChengXiang Zhai. Instance weighting for domain adaptation in nlp. In *ACL*, volume 7, pages 264–271, 2007.

[72] Thorsten Joachims. A support vector method for multivariate performance measures. In *Proceedings of the 22nd international conference on Machine learning*, pages 377–384. ACM, 2005.

[73] Lawrence Joseph, Theresa W Gyorkos, and Louis Coupal. Bayesian estimation of disease prevalence and the parameters of diagnostic tests in the absence of a gold standard. *American Journal of Epidemiology*, 141(3):263–272, 1995.

[74] Takafumi Kanamori, Shohei Hido, and Masashi Sugiyama. A least-squares approach to direct importance estimation. *Journal of Machine Learning Research*, 10(Jul): 1391–1445, 2009.

[75] George Karypis, Eui-Hong Han, and Vipin Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75, 1999.

[76] Hideko Kawakubo, Marthinus Christoffel Du Plessis, and Masashi Sugiyama. Computationally efficient class-prior estimation under class balance change using energy distance. *IEICE TRANSACTIONS on Information and Systems*, 99(1):176–186, 2016.

[77] Mark G Kelly, David J Hand, and Niall M Adams. The impact of changing populations on classifier performance. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 367–371. ACM, 1999.

[78] Daniel Kifer, Shai Ben-David, and Johannes Gehrke. Detecting change in data streams. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 180–191. VLDB Endowment, 2004.

[79] Sotiris Kotsiantis, Dimitris Kanellopoulos, Panayiotis Pintelas, et al. Handling imbalanced datasets: A review. *GESTS International Transactions on Computer Science and Engineering*, 30(1):25–36, 2006.

[80] Elizaveta Levina and Peter Bickel. The earth mover's distance is the mallows distance: Some insights from statistics. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 2, pages 251–256. IEEE, 2001.

[81] Paul S Levy and Edward H Kass. A three-population model for sequential screening for bacteriuria. *American Journal of Epidemiology*, 91(2):148–154, 1970.

[82] Mingsheng Long, Jianmin Wang, Guiguang Ding, Jiaguang Sun, and Philip S Yu. Transfer joint matching for unsupervised domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1410–1417, 2014.

[83] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. Learning transferable features with deep adaptation networks. In *International Conference on Machine Learning*, pages 97–105, 2015.

[84] Yishay Mansour, Mehryar Mohri, and Afshin Rostamizadeh. Domain adaptation with multiple sources. In *Advances in neural information processing systems*, pages 1041–1048, 2009.

[85] Locksley L McV Messam, Adam J Branscum, Michael T Collins, and Ian A Gardner. Frequentist and bayesian approaches to prevalence estimation using examples from johne's disease. *Animal Health Research Reviews*, 9(01):1–23, 2008.

[86] Letizia Milli, Anna Monreale, Giulio Rossetti, Fosca Giannotti, Dino Pedreschi, and Fabrizio Sebastiani. Quantification trees. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pages 528–536. IEEE, 2013.

[87] Letizia Milli, Anna Monreale, Giulio Rossetti, Dino Pedreschi, Fosca Giannotti, and Fabrizio Sebastiani. Quantification in social networks. In *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on*, pages 1–10. IEEE, 2015.

[88] Jose G Moreno-Torres, Troy Raeder, Rocío Alaiz-Rodríguez, Nitesh V Chawla, and Francisco Herrera. A unifying view on dataset shift in classification. *Pattern Recognition*, 45(1):521–530, 2012.

[89] Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006.

[90] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[91] Sinno Jialin Pan, James T Kwok, and Qiang Yang. Transfer learning via dimensionality reduction. In *AAAI*, volume 8, pages 677–682, 2008.

[92] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks*, 22 (2):199–210, 2011.

[93] Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. *Dataset shift in machine learning*. The MIT Press, 2009.

[94] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y Ng. Self-taught learning: transfer learning from unlabeled data. In *Proceedings of the 24th international conference on Machine learning*, pages 759–766. ACM, 2007.

[95] Walter J Rogan and Beth Gladen. Estimating prevalence from the results of a screening test. *American journal of epidemiology*, 107(1):71–76, 1978.

[96] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99–121, 2000.

[97] Marco Saerens, Patrice Latinne, and Christine Decaestecker. Adjusting the outputs of a classifier to new a priori probabilities: a simple procedure. *Neural computation*, 14(1):21–41, 2002.

[98] Dino Sejdinovic, Bharath Sriperumbudur, Arthur Gretton, and Kenji Fukumizu. Equivalence of distance-based and rkhs-based statistics in hypothesis testing. *The Annals of Statistics*, pages 2263–2291, 2013.

[99] Thomas A Severini. *Elements of distribution theory*, volume 17. Cambridge University Press, 2005.

[100] Jian Shen, Yanru Qu, Weinan Zhang, and Yong Yu. Wasserstein distance guided representation learning for domain adaptation. 2018.

[101] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.

[102] Melvin Dali Springer. The algebra of random variables. Technical report, 1979.

[103] Amos Storkey. When training and test sets are different: characterizing learning transfer. *Dataset shift in machine learning*, pages 3–28, 2009.

[104] Masashi Sugiyama and Motoaki Kawanabe. *Machine learning in non-stationary environments: Introduction to covariate shift adaptation*. MIT Press, 2012.

[105] Masashi Sugiyama, Shinichi Nakajima, Hisashi Kashima, Paul V Buenau, and Motoaki Kawanabe. Direct importance estimation with model selection and its application to covariate shift adaptation. In *Advances in neural information processing systems*, pages 1433–1440, 2008.

[106] Shiliang Sun, Honglei Shi, and Yuanbin Wu. A survey of multi-source domain adaptation. *Information Fusion*, 24:84–92, 2015.

[107] Zhaonan Sun, Nawanol Ampornpunt, Manik Varma, and Svn Vishwanathan. Multiple kernel learning and the smo algorithm. In *Advances in neural information processing systems*, pages 2361–2369, 2010.

[108] Gábor J Székely and Maria L Rizzo. Energy statistics: A class of statistics based on distances. *Journal of statistical planning and inference*, 143(8):1249–1272, 2013.

[109] Dirk Tasche. Does quantification without adjustments work? *arXiv preprint arXiv:1602.08780*, 2016.

[110] Antonio Torralba and Alexei A Efros. Unbiased look at dataset bias. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1521–1528. IEEE, 2011.

[111] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the twenty-first international conference on Machine learning*, page 104. ACM, 2004.

[112] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep domain confusion: Maximizing for domain invariance. *arXiv preprint arXiv:1412.3474*, 2014.

[113] Eric Tzeng, Judy Hoffman, Trevor Darrell, and Kate Saenko. Simultaneous deep transfer across domains and tasks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4068–4076, 2015.

[114] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. *arXiv preprint arXiv:1702.05464*, 2017.

[115] Vladimir Naumovich Vapnik. An overview of statistical learning theory. *IEEE transactions on neural networks*, 10(5):988–999, 1999.

[116] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[117] Slobodan Vucetic and Zoran Obradovic. Classification on data with biased class distribution. In *Machine Learning: ECML 2001*, pages 527–538. Springer, 2001.

[118] Geoffrey I Webb and Kai Ming Ting. On the application of roc analysis to predict classification performance under varying class distributions. *Machine learning*, 58 (1):25–32, 2005.

[119] Kilian Q Weinberger. Marginalized stacked denoising autoencoder. `http://www.cs.cornell.edu/~kilian/code/code.html`, 2018. [Online; accessed 14-August-2018].

[120] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83, 1945.

[121] Jack Chongjie Xue and Gary M Weiss. Quantification and semi-supervised classification methods for handling changes in class distribution. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 897–906. ACM, 2009.

[122] Bianca Zadrozny. Learning and evaluating classifiers under sample selection bias. In *Proceedings of the twenty-first international conference on Machine learning*, page 114. ACM, 2004.

[123] Mohammed J Zaki, Markus Peters, Ira Assent, and Thomas Seidl. Clicks: An effective algorithm for mining subspace clusters in categorical datasets. *Data & Knowledge Engineering*, 60(1):51–70, 2007.

[124] Kun Zhang, Bernhard Schölkopf, Krikamol Muandet, and Zhikun Wang. Domain adaptation under target and conditional shift. In *ICML (3)*, pages 819–827, 2013.

[125] Xu Zhang, Felix Xinnan Yu, Shih-Fu Chang, and Shengjin Wang. Deep transfer network: Unsupervised domain adaptation. *arXiv preprint arXiv:1503.00591*, 2015.

[126] Zhi-Hua Zhou. A brief introduction to weakly supervised learning. *National Science Review*, 5(1):44–53, 2017.

[127] Fuzhen Zhuang, Xiaohu Cheng, Ping Luo, Sinno Jialin Pan, and Qing He. Supervised representation learning: Transfer learning with deep autoencoders. In *IJCAI*, pages 4119–4125, 2015.

Appendix A

Quantification methods

A.1 Forman's *Adjusted Count* as matrix-inversion

The matrix-inversion *classify and adjust* method is set out in Section 3.2. The estimate of counts by class in the test set $\hat{\mathbf{a}}_t$ is given by:

$$\hat{\mathbf{a}_t} = \mathrm{R}_v^{-1}\mathbf{p}_t, \tag{A.1}$$

where $\hat{\mathbf{a}}$ is the vector of estimated counts by class:

$$\hat{\mathbf{a}}_t = \begin{pmatrix} \hat{a}_{t0} \\ \hat{a}_{t1} \end{pmatrix}, \tag{A.2}$$

and $\hat{a}_0$ and $\hat{a}_1$ are the estimated count of instances by class 0 and 1 respectively and where $\mathbf{p}_t$ is the vector of predicted counts by class for the test set as given by the classifier:

$$\mathbf{p}_t = \begin{pmatrix} p_{t0} \\ p_{t1} \end{pmatrix}, \tag{A.3}$$

R was defined in Equation 3.6 as:

$$\mathrm{R} = \begin{pmatrix} r_0 & (1-r_1) \\ (1-r_0) & r_1 \end{pmatrix}. \tag{A.4}$$

Adding the sub-scripts to designate the validation set and inverting gives:

$$\mathrm{R}_v^{-1} = \frac{1}{r_{v0}r_{v1} - (1-r_{v0})(1-r_{v1})} \begin{pmatrix} r_{v1} & (r_{v1}-1) \\ (r_{v0}-1) & r_{v0} \end{pmatrix}, \tag{A.5}$$

so $\hat{a}_0$ is:

$$\hat{a}_0 = \frac{r_{v1}p_{t0} + (r_{v1}-1)p_{t1}}{r_{v0}r_{v1} - (1-r_{v0})(1-r_{v1})}. \tag{A.6}$$

The test set contains n_t instances so:

$$n_t = a_{t0} + a_{t1} = p_{t0} + p_{t1}. \tag{A.7}$$

If we define the *proportion* of each *actual* class i in the test set as m_i where:

$$m_i = \frac{a_i}{n_t}, \tag{A.8}$$

and the proportion of each *predicted* class i in the test set as q_i where:

$$q_i = \frac{p_i}{n_t}, \tag{A.9}$$

then our estimate of the proportion of class 0 in the test set, $\hat{m}_0$ is given by:

$$\hat{m}_0 = \frac{r_{v1}q_0 + (r_{v1} - 1)q_1}{r_{v0}r_{v1} - (1 - r_{v0})(1 - r_{v1})}, \tag{A.10}$$

which simplifies to:

$$\hat{m}_0 = \frac{q_0 - (1 - r_{v1})}{r_{v0} - (1 - r_{v1})}. \tag{A.11}$$

By definition, the *true positive rate* for class 0, tpr_0, and the *false positive rate* for class 0, fpr_0, computed with the validation set are given by:

$$tpr_0 = r_{v0}, \tag{A.12}$$

and:

$$fpr_0 = 1 - r_{v1}, \tag{A.13}$$

so the estimate of the proportion of class 0 in the set, $\hat{m}_0$ is given by:

$$\hat{m}_0 = \frac{q_0 - fpr_0}{tpr_0 - fpr_0}, \tag{A.14}$$

which is the *Adjusted Count* formula from Forman [43].

A.2 Saerens et al. [97] probabilistic expectation-maximisation method

The method relies on a classifier that generates an output that can be interpreted as the probability of an instance belonging to a class. The adjustment method takes the class probabilities assigned to each instance in the test set by the trained classifier as its input and generates both an estimate of the class distribution in the test set and revised class probabilities for each instance.

With Bayes rule, the estimate of the probability of the data $\mathbf{x}$ given the class label y_i for our training set, $\hat{P}_{tr}(\mathbf{x}|y_i)$, can be expressed as:

$$\hat{P}_{tr}(\mathbf{x}|y_i) = \frac{\hat{P}_{tr}(y_i|\mathbf{x})\hat{P}_{tr}(\mathbf{x})}{\hat{P}_{tr}(y_i)}. \tag{A.15}$$

Similarly for the test set:

$$\hat{P}_{te}(\mathbf{x}|y_i) = \frac{\hat{P}_{te}(y_i|\mathbf{x})\hat{P}_{te}(\mathbf{x})}{\hat{P}_{te}(y_i)}. \tag{A.16}$$

The usual assumption is made that the class-conditional feature distribution is the same in the training and the test set i.e.:

$$\hat{P}_{tr}(\mathbf{x}|y_i) = \hat{P}_{te}(\mathbf{x}|y_i), \tag{A.17}$$

so the posteriori probabilities in the test set can be given by:

$$\hat{P}_{te}(y_i|\mathbf{x}) = \frac{\frac{\hat{P}_{te}(y_i)}{\hat{P}_{tr}(y_i)}\hat{P}_{tr}(y_i|\mathbf{x})}{\sum_{j=1}^{n}\frac{\hat{P}_{te}(y_j)}{\hat{P}_{tr}(y_j)}\hat{P}_{tr}(y_j|\mathbf{x})}. \tag{A.18}$$

Clearly this relies on knowing, or estimating, the class distribution (the class priors) in the test set which of course is the quantification problem that we are trying to solve. The class priors are estimated with an *Expectation-Maximisation* algorithm:

Consider the test set to consist of n items $(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)$ and that there are c classes.

The likelihood of seeing this set of data is given by:

$$L(\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_n) = \prod_{k=1}^{n} P_{te}(\mathbf{x}_k) \tag{A.19}$$

$$= \prod_{k=1}^{n}\left[\sum_{i=1}^{c} P_{te}(\mathbf{x}_k, y_i)\right] \tag{A.20}$$

$$= \prod_{k=1}^{n}\left[\sum_{i=1}^{c} P_{te}(\mathbf{x}_k|y_i)P_{te}(y_i)\right]. \tag{A.21}$$

Again, making the within-class invariance assumption that $P_{tr}(\mathbf{x}_k|y_i) = P_{te}(\mathbf{x}_k|y_i)$ the EM algorithm then estimates the class priors for the new data $\hat{P}_{te}(y_i)$ that maximise the likelihood of the data.

$\hat{P}_{tr}(y_i|\mathbf{x}_k)$ is the output of the model, trained on the training set, corresponding to class y_i when the input data is $\mathbf{x}_k$.

$\hat{P}_{tr}(y_i)$ is the class prior at training time and is simply given by the class count in the training set.

In the EM method both $\hat{P}_{te}(y_i)$ and $\hat{P}_{te}(y_i|\mathbf{x}_k)$ are re-estimated on each iteration. The superscript s being used to denote the iteration step.

The class prior is initialised to the value from the training set:

$$\hat{P}^0_{te} = \hat{P}_{tr}(y_i). \tag{A.22}$$

The two steps at each iteration are then:

1. Update the posteriori probability based on the estimated class priors:

$$\hat{P}^s_{te}(y_i|\mathbf{x}_k) = \frac{\frac{\hat{P}^s_{te}(y_i)}{\hat{P}_{tr}(y_i)}\hat{P}_{tr}(y_i|\mathbf{x}_k)}{\sum_{j=1}^{c}\frac{\hat{P}^s_{te}(y_j)}{\hat{P}_{tr}(y_j)}\hat{P}_{tr}(y_j|\mathbf{x}_k)}. \tag{A.23}$$

2. Update the estimate of the class prior probabilities by summing the a posteriori estimates:

$$\hat{P}^{s+1}_{te}(y_i) = \frac{1}{N}\sum_{k=1}^{N}\hat{P}^s_{te}(y_i|\mathbf{x}_k). \tag{A.24}$$

The iterations continue until a convergence criteria is met.

A.3 Joachims [72] SVM for multivariate performance measures

Instead of mapping *individual* instance mapping feature vectors to estimated labels the $\text{SVM}^{\Delta}_{multi}$ hypothesis $\bar{h}$ maps the tuple of feature vectors to the tuple of estimated labels:

$$\bar{y} = \bar{h}_{\mathbf{w}}(\bar{\mathbf{x}}) \tag{A.25}$$

Where $\bar{\mathbf{x}}$ is a tuple of n feature vectors $(\mathbf{x}_1, ..., \mathbf{x}_n)$ and $\bar{y}$ is the corresponding tuple of n labels $(y_1, ..., y_n)$ and $\bar{y}'$ is the corresponding tuple of n estimated labels $(y'_1, ..., y'_n)$.

Joachims [72] SVM optimisation problem is:

$$\min_{\mathbf{w},\xi\geq 0}\frac{1}{2}\mathbf{w}.\mathbf{w} + C\xi \tag{A.26}$$

Such that:

$$\forall\bar{y}' \in \bar{\mathcal{Y}} \setminus \bar{y} : \mathbf{w}^T[\Psi(\bar{\mathbf{x}}, \bar{y}) - \Psi(\bar{\mathbf{x}}, \bar{y}')] \geq \Delta(\bar{y}', \bar{y}) - \xi \tag{A.27}$$

Where: $\mathcal{Y} = \{-1, +1\}$

$\Psi(,)$ is simply:

$$\Psi(\bar{\mathbf{x}}, \bar{y}) = \sum_{i=1}^{n} y_i' \mathbf{x}_i \tag{A.28}$$

$\Delta(\bar{y}', \bar{y})$ is the loss function between the tuple of actual labels and the tuple of estimated labels. This allows a loss to be computed over the elements of a confusion matrix for example.

A.4 Hofer [62] distribution matching with Gaussian mixtures

The conditional distributions in the Source domains and the unconditional distribution in the Target domain are considered to be separate mixtures of sub-distributions. Hofer [62] states that the model is not restricted to a particular mixture distribution but that they chose a Gaussian mixture.

In the Source domain the class-conditional distributions, $f_S(x|y)$ are considered to be a weighted sum of K_y Gaussian distributions $h_i(x|y)$:

$$f_S(x|y) = \sum_{i=1}^{K_y} \alpha_{iy} h_i(x|y) \qquad y = 0, 1. \tag{A.29}$$

The unconditional distribution is always simply the sum of the class-conditional distributions weighted by the class proportions i.e.:

$$f(x) = P(y = 0) f(x|y = 0) + P(y = 1) f(x|y = 1). \tag{A.30}$$

The unconditional distribution in the Target domain $f_T(x)$ is also considered to be a weighted sum of Gaussian distributions. In this case the weighted sum of L distributions $g_j(x)$:

$$f_T(x) = \sum_{j=1}^{L} \gamma_j g_j(x). \tag{A.31}$$

The relationship between the distributions $h_i(x|y)$ and the distributions $g_j(x)$ is given by the matrices M_0 and M_1. The elements of the matrix M_y, m_{ijy}, describe the proportion of the distribution $h_i(x|y)$ that is within $g_j(x)$.

In order for probability mass to be conserved, clearly:

$$\sum_{j=1}^{L} m_{ijy} = 1 \qquad \forall i = 1, ..., K_y \qquad y = 0, 1. \tag{A.32}$$

So:

$$\gamma_j = \sum_{i=1}^{K_0} P_T(y=0)\alpha_{i0}m_{ij0} + \sum_{i=1}^{K_1} P_T(y=1)\alpha_{i1}m_{ij1}. \tag{A.33}$$

This is rewritten as:

$$\gamma_j = \sum_{i=1}^{K_0+K_1} a_{ij}, \tag{A.34}$$

where:

$$a_{ij} = \begin{cases} P_T(y=0)\alpha_{i0}m_{ij0} & i = 1, ..., K_0 \\ P_T(y=1)\alpha_{i1}m_{ij1} & i = 1, ..., K_1. \end{cases} \tag{A.35}$$

With only two classes, 0 and 1:

$$P_S(y=0) + P_S(y=1) = P_T(y=0) + P_T(y=1) = 1 \tag{A.36}$$

The parameters for the two class-conditional Gaussian mixtures from the Source domain $f_S(x|y=0)$ and $f_S(x|y=1)$, and the unconditional Gaussian mixture from the Target domain $f_T(x)$ are all estimated separately from the labelled Source data and unlabelled Target data using the standard EM algorithm for Gaussian mixtures. This gives estimates for the parameters $\{\alpha_{10}, \alpha_{20}, ..., \alpha_{K_0 0}\}$, $\{\alpha_{11}, \alpha_{21}, ..., \alpha_{K_1 1}\}$ and $\{\gamma_1, \gamma_2, ..., \gamma_L\}$ along with the parameters for the Gaussian distributions $\{h_1(x|y=0), h_2(x|y=0), ..., h_{K_0}(x|y=0)\}$, $\{h_1(x|y=1), h_2(x|y=1), ..., h_{K_1}(x|y=1)\}$ and $\{g_1(x), g_2(x), ..., h_L(x)\}$.

Hofer [62] does not give details on how number of components in each mixture (K_0, K_1, L) are set.

The second step is to estimate the other model parameters: the elements of the M matrices and the class distribution in the Target domain $P_T(y)$.

The full set of parameters enables values of $\{\gamma_1, \gamma_2, ..., \gamma_L\}$ to be recalculated using Equation A.33 and then for the unconditional distribution to be synthesised with these parameter values using Equation A.31). This is done with a two step iterative process where at each step the Earth Mover Distance (see Section 2.2.3.3) is minimised.

Earth Mover Distance (EMD) is given by:

$$\text{EMD} = \sum_{i=1}^{K_0+K_1} \sum_{j=1}^{L} a_{ij} d_{ij}, \tag{A.37}$$

where d_{ij} is the Euclidean distance between the centroid of the distribution h_i and the centroid of the distribution g_j and a_{ij} represents the mass transferred.

The two steps of this iterative stage of the process are:

1. Minimise EMD with respect to $P_T(y)$ holding the M parameters constant
2. Minimise EMD with respect to the M parameters holding $P_T(y)$ constant

While Hofer [62] use Euclidean distance they make it clear that the choice of distance measure is not pre-determined and that cross-validation can be used to select a measure. In practice an adjustment was made to the Euclidean distance values d_{ij} to avoid unwanted splitting of components where some components move large distances. They replaced d_{ij} with d'_{ij} where:

$$d'_{ij} = d_{ij} + \psi f_{ij}, \tag{A.38}$$

where:

$$f_{ij} = \begin{cases} |\gamma_j - P_T(y=0)\alpha_{i0}| & i = 1, ..., K_0 \\ |\gamma_j - P_T(y=1)\alpha_{i1}| & i = K_0, ..., K_0 + K_1. \end{cases} \tag{A.39}$$

The value of ψ was found using cross-validation.

At this stage the estimate for the class distribution in the Target domain, $P_T(y)$, has been made. The authors then go on the use the estimated parameters to generate estimates of the class-conditional distributions in the Target domain.

Appendix B

Importance weighting methods

B.1 Importance weighting methods generally

The standard Empirical Risk Minimisation (ERM) framework for supervised learning is [70]:

$$\theta^* = \arg\min_{\theta \in \Theta} \sum_{(x,y)\in(X,Y)} P(x,y) l(x,y,\theta) \tag{B.1}$$

Where θ^* is our optimal model and $l(x,y,\theta)$ is our chosen loss function.

The aim is to have an optimal model for the *Target* domain i.e.:

$$\theta_T^* = \arg\min_{\theta \in \Theta} \sum_{(x,y)\in(X,Y)} P_T(x,y) l(x,y,\theta) \tag{B.2}$$

However the training instances have been randomly sampled from the *Source* domain so if we rewrite equation B.3 as:

$$\theta_T^* = \arg\min_{\theta \in \Theta} \sum_{(x,y)\in(X,Y)} \frac{P_T(x,y)}{P_S(x,y)} P_S(x,y) l(x,y,\theta) \tag{B.3}$$

$P_S(X,Y)$ is unknown so we estimate it from the empirical distribution $P_S^e(X,Y)$ and now our estimate of θ_T^* becomes:

$$\hat{\theta}_T^* = \arg\min_{\theta \in \Theta} \sum_{(x,y)\in(X,Y)} \frac{P_T(x,y)}{P_S(x,y)} P_S^e(x,y) l(x,y,\theta) \tag{B.4}$$

If our set of training data from the Source domain D_S consists of N_S instances $\{(x_{Si}, y_{Si})\}$ then we can rewrite equation B.4 as:

$$\hat{\theta}_T^* = \arg\min_{\theta \in \Theta} \sum_{i=1}^{N_S} \frac{P_T(x_{Si}, y_{Si})}{P_S(x_{Si}, y_{Si})} l(x_{Si}, y_{Si}, \theta) \tag{B.5}$$

i.e. each instance i of data in the training set is weighted by weight w_i:

$$w_i = \frac{P_T(x_{Si}, y_{Si})}{P_S(x_{Si}, y_{Si})} \tag{B.6}$$

However, our true distributions $P_T(\mathbf{x}, y)$ and $P_S(\mathbf{x}, y)$ and unknown.

B.2 Kernel mean matching

$$\hat{\mathbf{w}} = \{\hat{w}_1, \hat{w}_2, ...\hat{w}_{N_S}\} \tag{B.7}$$

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \left(\frac{1}{2}\mathbf{w}^T K \mathbf{w} - \kappa^T \mathbf{w}\right) \tag{B.8}$$

Under certain constraints of:

$$w_i \in [0, B], \tag{B.9}$$

and

$$\left| \sum_{i=1}^{N_S} w_i - N_S \right| \leq N_S \epsilon. \tag{B.10}$$

Where $\mathbf{w}$ is the vector of weights for the N_S instances of training data from the Source domain:

$$\mathbf{w} = w_1, ..., w_{N_S} \tag{B.11}$$

and

$$K = \begin{pmatrix} K_{S,S} & K_{S,T} \\ K_{T,S} & K_{T,T} \end{pmatrix} \tag{B.12}$$

and

$$K_{ij} = k(x_i, x_j) \tag{B.13}$$

and

$$\kappa_i = \frac{N_S}{N_T} \sum_{j=1}^{N_T} k(x_i, x_{T_j}) \tag{B.14}$$

where

$$x_i \in X_S \cup X_T \tag{B.15}$$

and

$$x_{T_j} \in X_T \tag{B.16}$$

B.3 Unconstrained least squares importance fitting

As before the importance weights are defined as the ratio of the probability densities for that point x in the Target and in the Source domains:

$$w_i = \frac{P_T(x_{Si})}{P_S(x_{Si})} \tag{B.17}$$

Where x_{Si} is the i^{th} datapoint in the training set from the Source domain.

Firstly the squared loss l_{SQ} is defined:

$$l_{SQ} = \int \big(\hat{w}(x) - w(x)\big)^2 P_S(x)dx \tag{B.18}$$

Expanding the squared term:

$$l_{SQ} = \int \hat{w}(x)^2 P_S(x)dx - 2\int \hat{w}(x)w(x)P_S(x)dx + \int w(x)^2 P_S(x)dx \tag{B.19}$$

As we will minimise the loss function with respect to $\hat{w}(x)$ the final term is constant and so can be ignored in the minimisation:

$$\int w(x)^2 P_S(x)dx = C \tag{B.20}$$

The loss function now becomes:

$$l_{SQ} = \int \hat{w}(x)^2 P_S(x)dx - 2\int \hat{w}(x)w(x)P_S(x)dx + C \tag{B.21}$$

From our definition of w:

$$w(x)P_S(x) = P_T(x) \tag{B.22}$$

So the loss function becomes:

$$l_{SQ} = \int \hat{w}(x)^2 P_S(x)dx - 2\int \hat{w}(x)P_T(x)dx + C \tag{B.23}$$

The loss function can be approximated by using the empirical values from the training set (Source) and test set (Target):

$$\hat{l}_{SQ} = \frac{1}{N_S}\sum_{j=1}^{N_S} \hat{w}(x_{Sj})^2 - \frac{2}{N_T}\sum_{i=1}^{N_S} \hat{w}(x_{Ti}) + C \tag{B.24}$$

The estimated weight function $\hat{w}(\mathbf{x})$ is modelled as a sum of b basis functions $\phi(\mathbf{x})$ weighted with the constants α:

$$\hat{w}(\mathbf{x}) = \sum_{l=1}^{b} \alpha_l \phi_l(\mathbf{x}) = \boldsymbol{\alpha}^T \boldsymbol{\phi}(\mathbf{x}) \tag{B.25}$$

So the loss function can be expressed as:

$$l_{SQ} = \boldsymbol{\alpha}^T \hat{\mathrm{H}} \boldsymbol{\alpha} - 2\hat{h}^T \boldsymbol{\alpha} \tag{B.26}$$

Where:

$$\hat{\mathrm{H}} = \frac{1}{N_S} \sum_{j=1}^{N_S} \phi(x_{Sj}) \phi(x_{Sj})^T \tag{B.27}$$

And:

$$\hat{h} = \frac{1}{N_T} \sum_{i=1}^{N_T} \phi(x_{Ti}) \tag{B.28}$$

With l_2 regularisation the constraint that the values of α must all be positive can be dropped and hence this is *unconstrained*. This now gives the unconstrained minimisation:

$$\min_{\boldsymbol{\alpha}} \left[\frac{1}{2} \boldsymbol{\alpha}^T \hat{\mathrm{H}} \boldsymbol{\alpha} - \hat{h}^T \boldsymbol{\alpha} + \frac{\lambda}{2} \boldsymbol{\alpha}^T \boldsymbol{\alpha} \right] \tag{B.29}$$

Which has a closed form solution:

$$\tilde{\boldsymbol{\alpha}} = (\hat{\mathrm{H}} + \lambda \mathrm{I})^{-1} \hat{h} \tag{B.30}$$

Since the non-negativity constraint on α was dropped it is possible that come of the learned values are negative so to compensate for this approximation error the solution is modified as:

$$\hat{\boldsymbol{\alpha}} = \max(0, \tilde{\boldsymbol{\alpha}}) \tag{B.31}$$

********** End of document **********