The effectiveness of subspace mapping techniques adapted to unlabeled samples from a global domain in mitigating sample selection bias

Timo van Hoorn

Supervisor(s): Joana de Pinho Gonçalves, Yasin Tepeli

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2023

Name of the student: Timo van Hoorn
Final project course: CSE3000 Research Project
Thesis committee: Joana de Pinho Gonçalves, Yasin Tepeli, Julian Urbano Merino

Abstract

Sample selection bias occurs when the selected samples in a subset of the original data set follow a different distribution than the samples from the original data set. This type of bias in the training set could result in a classifier being unable to predict samples from a testing data set optimally. Domain adaptation techniques try to adapt classifiers to a possible bias in the training or testing set. Subspace mapping techniques specifically do this by trying to find common subspaces between the source and target domain, where the source domain is the domain with all samples used for training, and the target domain is the domain with samples that must be predicted. This project aims to evaluate the effectiveness of two subspace mapping techniques in mitigating sample selection bias. This research assumes that no data samples from a target domain are available, but only unlabelled samples coming from an underlying global domain. The two subspace mapping techniques that will be tested in this paper are subspace alignment (SA) and transfer component analysis (TCA). This paper will show that the subspace alignment method is more effective on data sets with fewer features and where the source and target domains are further away from each other. The transfer component analysis method is more effective when more training samples are available on data sets with fewer features and where the distance between the source and target domain is not too big. The effectiveness of both methods also depends on the type and form of the data sets they are used on.

1 Introduction

In machine learning, we speak of *sample selection bias* when some samples with specific feature values appear more in the selected samples from the original data set than when they are selected completely randomly. When training or testing data has this specific bias, the source and target domain can differ, where the *source domain* is the domain with all samples used for training, and the *target domain* is the domain with samples that must be predicted. A difference between the source and target domain could lead to worse predictions of the samples in the target domain, as most classifiers do not consider this difference. Mitigating sample selection bias can help machine learning models become more accurate and reliable. Therefore, it is worth researching techniques that take biased data into account.

Some studies have already been done on sample selection bias. One idea already studied is the effect of sample selection bias on the results of some well-known classifiers [9]. Some papers also discuss different methods to mitigate the problem of selection bias [9] [7]. These methods are called *domain adaptation methods*. Domain adaptation methods try to mitigate the negative effects of bias in the data by adapting the source domain to the target domain. Wouter M. Kouw and Marco Loog [4] divide domain adaption methods into sample, feature, and inference-based methods. First of all, sample-based strategies focus on the process of selecting samples in a given data set. Secondly, feature-based techniques will use transformations in the feature space of the data sets to generalize the data. Lastly, inference-based methods use adaptation in the parameter estimation procedure in classification techniques. One specific category of feature-based approaches is *subspace mapping*. Subspace mapping techniques aim to decrease the shift between the source and target domains by finding common subspaces between the two domains. Using these subspaces, the source data will be aligned with the target data [4].

Most papers on domain adaptation techniques test the effectiveness of their methods by using the target domain to adapt their models. No data from the target domain is sometimes available, but unlabeled data from an underlying global domain is. In these cases, it is possible to use unlabeled data from the underlying global domain to generalize classifiers instead of using samples from a target domain, where the *global domain* is the domain that contains all possible data samples for a specific problem. Samples from the source and target domain always lie in this global domain. There is still a lack of research on the effectiveness of using unlabeled data from a global domain to adapt the source domain to the target domain. This also applies to subspace mapping techniques.

In this paper, we investigate the effectiveness of subspace mapping techniques in mitigating sample selection bias in classification problems, assuming that no samples from the target domain are available and samples from an underlying global domain are. These subspace mapping techniques are examined with four different factors that could influence their effectiveness. Firstly, we explore the impact of training sample size on their effectiveness. Secondly, we look at how the number of features influences their performance. Thirdly, we investigate the influence of the distance between the source and target domains on their effectiveness. Lastly, we examine the effect of different data sets on their performance. By investigating these four factors, we aim to provide insights into the overall effectiveness of subspace mapping techniques in mitigating sample selection bias. The two specific subspace mapping techniques that will be researched in this paper are *subspace alignment* (SA) [2] and *transfer component analysis* (TCA) [5]. The reasons for choosing these two methods are that different papers support and discuss the implementation of both methods [2] [5], both methods are already implemented in a library, and both methods differ in their approach to mapping the source to the target domain.

In Section 2, the methods used in this research will be explained. Section 3 will give and discuss the results for all experiments. The research's ethical aspects and reproducibility will be explained in Section 4. Lastly, Section 5 will give the conclusion and suggest topics for future research.

2 Methodology

This section outlines the methods used to answer the research question. Section 2.1 describes the setup of the data sets used in the research. After that, Section 2.2 describes the tech-

niques used to bias data sets. The implementation of all classifiers is described in Section 2.3. Section 2.5 will explain the proxy A-distance used as a distance metric for one of the experiments. Lastly, the evaluation criterion is described in Section 2.4.

2.1 Setup of the data sets

The scikit-learn data set library [6] was used to generate data samples. This library contains both real-world data sets and can create synthetic data sets. We chose to use synthetic data sets in our research, as synthetic data sets are easy to toy around with. Most data set generation methods in the scikit-learn library have a lot of parameters that can be changed, such as the number of features and the number of samples in the generated data sets, making it easy to set up different data sets to research on.

The data generation functions that were used in our research were the *make classification*, *make blobs*, and *make gaussian quantiles* functions. These functions are all part of the scikit-learn library. The *make classification* function creates a random data set that can be used for classification problems. The function has a lot of parameters that determine what the output data set looks like. These parameters are the number of samples, *n_samples*, the number of features, *n_features*, the number of classes, *n_classes*, the number of informative features, *n_informative*, the number of redundant features, *n_redundant*, the number of repeated features, *n_repeated*, the number of clusters per class, *n_cluster_per_class*, and the random state, *random_state*. There are even more parameters for this function, but since they will always be set to their default values in our experiments, they won't be mentioned.

The *make blobs* function generates a data set containing isotropic Gaussian blobs. The parameters for this function include the number of samples, *n_samples*, the number of features, *n_features*, the number of centers, *centers*, and the random state, *random_state*. There are some more parameters for the make blob function, but these will always be set to their default value. The number of classes that the data set generated by *make blobs* equals the value of *centers*. We want to work with data sets containing only two classes in our experiments. Therefore, we modified the data sets generated by *make blobs* to ensure only two classes will be present in the modified data set. To do this, we changed all class labels with an even value to 0 and all class labels with an odd value to 1. Like this, two classes remain with labels 0 and 1 when *centers* is set to a value higher than 2.

The *make gaussian quantiles* function generates an isotropic Gaussian and labels samples by quantile. The parameters for the function are the number of samples, *n_samples*, the number of features, *n_features*, the number of classes, *n_classes*, and the random state, *random_state*. Again, this function contains more parameters, but these will always be set to their default values.

A training, testing, and global data set must be obtained from a data set generated with one of the above functions to train and evaluate the classifiers. The training data set contains samples from the source domain, the testing data set contains samples from the target domain, and the global data set contains samples from the underlying global domain. The training, testing, and global data sets may not have duplicate samples. In the research, the testing and global data sets are both not biased. Therefore, both these data sets will contain random samples from the original data set. On the other hand, the training set should be biased in most of the test cases. Firstly, random samples will be selected from the original data set. Afterward, biased samples will be selected from these samples using the method described in Section 2.2. The reason for selecting random samples first is that directly selecting the biased samples from the original data set would make the remaining data samples biased. This would result in a biased testing and global data set, as these will be selected from the remaining samples. The ratio between the training, testing, and global data set was set to 5:2:3 for all experiments.

2.2 Biasing technique

As described in the previous Section, a biasing technique was applied to the training set to bias the training data. There are different ways to introduce sample selection bias into a data set [7] [9]. The biasing technique chosen for this research is based on a technique described in the Adapt Python library [1]. This technique randomly selects samples from a data set, with every sample having a probability of being chosen. However, how this probability gets calculated differs from the original technique in our research. In our method, a random data point gets generated that lies within the boundaries of the sample space. This generated data point has the same dimensions as the sample space. The next step is calculating the Euclidean distances between every sample and the generated data point. The Euclidean distance d is computed using Eq.1.

$$\mathrm{d}_{s,p} = \sqrt{\sum_{i=1}^{n}(x_{i,s} - x_{i,p})^2} \tag{1}$$

Where s is the sample, p is the generated data point, n is the number of features, $x_{i,s}$ is the value of feature i for sample s, and $x_{i,p}$ is the value of feature i for the generated data point p. The calculated distances are then used to calculate a selection weight for every sample. This selection weight w is calculated using Eq. 2.

$$\mathrm{w}_s = e^{\frac{(d_{s,p} \cdot -f)}{\sqrt{dim_s}}} \tag{2}$$

Where s is the sample, $d_{s,p}$ is the Euclidean distance between the sample s and the generated data point p, f is the biasing factor, which is either 0 or a positive integer, and dim_s is the number of dimensions of sample s. The weight of a sample decreases exponentially with a higher distance. This makes sure that samples close to the data point will have a higher weight than samples far away from the data point. The distance is multiplied by the negative value of the biasing factor to influence the weight increase when the distance changes. A higher biasing factor will result in a bigger weight difference between samples when distances remain the same. Using the negative value of the biasing factor will ensure that

samples with lower distances get a higher weight, given a positive value for the biasing factor. To get the final probability for a sample to get selected *p*, Eq. 3 is used.

$$\mathrm{p}_s = \frac{w_s}{\sum_{i=1}^{n} w_i} \tag{3}$$

Where s is the sample, w_s is the selection weight for sample s, n is the number of samples, and w_i the selection weight of sample i. The number of biased samples that will be selected from the unbiased data set x can be calculated using Eq. 4.

$$\mathrm{x} = r \cdot n \tag{4}$$

Where r is the ratio between selected samples and the total number of samples, and n is the total number of samples in the original, unbiased data set. Finally, The samples are selected using a *random choice* function from NumPy [3]. The *random choice* function has as parameters an input array, A, the number of items to select from the array, x, and an array of probabilities, P. The function outputs an array of size x containing samples of A that were selected with the probabilities belonging to P. In our case, the inputs are the original, unbiased data set containing all samples, the number of biased samples that are selected from the original data set (Eq. 4), and an array containing all calculated probabilities for samples (Eq. 3). The output is a biased subset of the unbiased input data set.

2.3 Domain adaptations methods

In our research, we used two different adaptation methods to experiment. The two adaptation methods were subspace alignment (SA) and transfer component analysis (TCA). the following two paragraphs will explain the two methods in a little more detail.

The idea behind the subspace alignment [2] is that a lower discrepancy between the source and target domain results in better predictions on the target domain by a classifier. The goal of the technique is to decrease the discrepancy between the source and target domain by finding shared subspaces between them and then aligning the source subspace with the target subspace. The first parameter of SA is the number of dimensions in the subspaces, *d*. The subspaces for the source and the target domains are obtained by applying principal component analysis (PCA). Using PCA will result in a subspace containing the *d* most important eigenvectors of the initial domain. After using PCA, the subspace of the source will be transformed using a matrix *M*. This matrix is constructed so that the result of multiplying *M* with the source subspace is as close as possible to the target subspace. This resulting target-aligned source subspace is then used to train an estimator. This estimator, *estimator*, is the second parameter of SA and can be any other classifier.

Transfer component analysis [5] is similar in some ways to subspace alignment. TCA also tries to bring the source and target domains closer through subspaces. However, the way that TCA does this differs from that of SA. TCA uses a kernel matrix *K* to calculate the distance between the source and target domains. Minimizing this distance will result in the kernel that represents the subspace for the target-aligned source domain. This subspace is then used to train an estimator. The parameters of TCA are the estimator used to train on the subspace, *estimator*, and the number of dimensions of the subspace, *d*. [1]

In our research, we used *logistic regression* (LR) and *k-nearest neighbors* (KNN) as *estimator* for SA and TCA. We chose these two classifiers as LR is a linear classifier, and KNN is a non-linear classifier. Researching both types of classifiers will give us more insight into the accuracies of the domain adaptation methods. [2]

Since parameters influence the performance of SA, TCA, and KNN, parameter optimization was performed for every evaluation of the classifiers. To get the best parameters possible for every data set, k-fold cross-validation was performed with the training data before fitting the models. K-fold cross-validation is an effective way to retrieve good values for the hyperparameters of classifiers. For SA and TCA, cross-validation was used to tune the dimensionality parameter, *d*. We chose not to fine-tune the other parameters of SA and TCA as that would exponentially increase the time to find the best parameters, and the experiment was already really time-consuming. For KNN, cross-validation was used to tune parameter *k*. This parameter represents the number of neighbors to consider when estimating a sample.

2.4 Evaluation criterion

All models were evaluated with a well-known accuracy score in machine learning. This accuracy score counts the number of samples evaluated correctly and divides that number by the total number of samples the model estimated. This accuracy score a has the following equation:

$$\mathrm{a} = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

Where TP = true positives, TN = true negatives, FP = false positives, and FN = false negatives. This accuracy score doesn't consider class imbalances. However, in the research, we ensured that each class in every data set contained equally many samples in the training, testing, and global data set. In this way, the class imbalance should not be corrected when calculating the accuracy. Equation 5 also does not consider class weights. This does not matter either since the research was done on synthetic rather than real-world data, and every incorrect class prediction should have the same effect on the model's effectiveness.

To decrease the influence of randomness on the results, we trained and evaluated every classifier 10 times for every data point, with the training, testing, global data set samples, and the biasing data point used in the biasing technique being different every time due to randomness. We displayed the mean accuracy, the minimum, and the maximum values in the results.

[1] The Python Adapt library [1] was used to implement SA and TCA.

[2] The scikit-learn library [6] was used to implement LR and KNN.

2.5 Proxy A-distance

For one of the experiments, subspace mapping techniques are evaluated against the distance between the source and the target domain. The distance metric that we chose for this experiment is the proxy A-distance. The method we used to calculate the proxy A-distance is based on two papers [8] [4]. The main idea behind the method is that for two similar source and target domains, it is harder to predict from which domain a random sample comes, given that the domain is unknown for the sample. Thus a classifier trained to predict whether a sample belongs to the source or target domain should give a bigger error when the two domains are more similar. To calculate the proxy A-distance, all samples from the source data set will be labeled 0, and all samples from the target data set will be labeled 1. The original labels for all the samples should be neglected and overwritten. The second step is mixing all samples from the source and target domains. After that, a classifier will be used to perform k-fold cross-validation on the mixed data set to get the cross-validation error. This error will be the mean of all errors for each fold. The error itself get calculated using Eq.6:

$$e = 1 - a \tag{6}$$

Where a is the accuracy in Eq.5. The classifier we used in our research was logistic regression, and we applied 5-fold cross-validation. The cross-validation error was then used in the following equation to calculate the proxy A-distance D:

$$\mathrm{D}_{S,T} = 2(1 - 2 \cdot E_{S,T}) \tag{7}$$

Where S is the data set in the source domain, T is the data set in the target domain, and $E_{S,T}$ is the cross-validation error obtained of a classifier trained to discriminate source samples in s from target samples in T like explained above.

3 Results and Discussion

This section will discuss the setup and results of all experiments. In all results, SA and TCA are compared to either KNN or LR depending on which classifier is used for *estimator*. Furthermore, every plot has an 'optimal' line representing the results of LR or KNN trained on an unbiased training set.

In the first experiment, the accuracy of all classifiers was tested on different numbers of training set samples. In the second experiment, accuracy was measured on different numbers of features. Accuracy was measured for different proxy-A distances between the source and target domain in the third experiment. Lastly, the fourth experiment measured the accuracy of different types of data sets.

3.1 Experiment 1: Training sample size

For the first set of experiments, the mean accuracies of SA and TCA were measured for different training sample sizes. The data set was generated using the *make classification* function (Section 2.1). We set the parameters to n_samples = 9000, *n_features* = 2, *n_classes* = 2, *n_informative* = 2, *n_redundant* = 0, *n_repeated* = 0, *n_clusters_per_class* = 2, and *random_state* = 0.

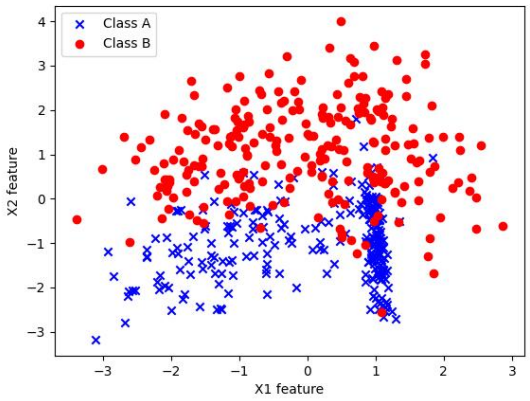


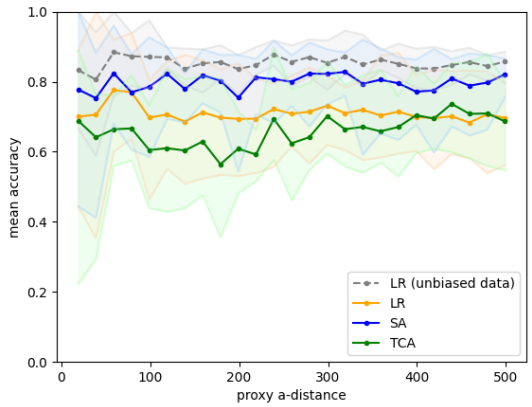
Figure 1: Data set generated by the *make classification* function. Parameters are set to n_samples = 500, *n_features* = 2, *n_classes* = 2, *n_informative* = 2, *n_redundant* = 0, *n_repeated* = 0, *n_clusters_per_class* = 2, and *random_state* = 0. The rest of the parameters were set to their default values

To generate data sets with different numbers of samples, a subset of samples was selected and drawn from the original data for each measurement. Figure 1 shows an example of the generated data set, where *n_samples* is set to 500. This data set was then split into training, testing, and global data sets. After that, the biasing technique explained in Section 2.2 was used to biasing the training set. The biasing factor f used in this experiment was set at 50, and the ratio between selected samples and the total number of samples r was set at 0.1.

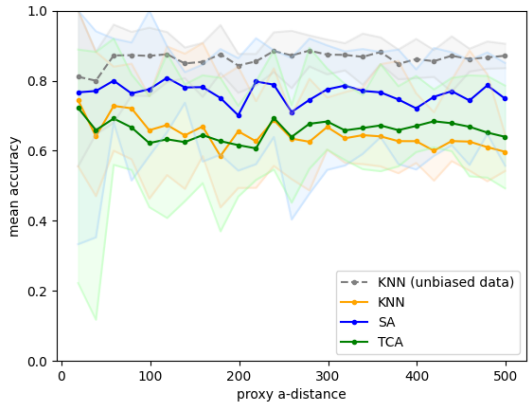
Figure 2 shows the results of the experiments using LR and KNN. In the experiment with Logistic Regression (Figure 2a), the mean accuracy of SA is better than LR and TCA for every training set sample size. However, for less than 60 training samples, the *range* for SA is bigger than the *range* of LR, where the range is the difference between the highest and lowest accuracy measured. The reason for this could be due to the complexity of the alignment process and the potential instability of the learned mappings when the sample size is small. TCA has a lower mean accuracy than LR at low training sample sizes. From 400 samples onwards, TCA and LR have around the same mean accuracies. In the KNN experiment (Figure 2b), SA also has better mean accuracy for every data point than KNN and TCA. For smaller training sample sizes, TCA performs similarly to KNN. From 300 samples, TCA's mean accuracies get better than those of KNN. TCA makes use of a kernel function in its method. The kernel better represents the source and global domains with more data samples. This should give a more accurate result when trying to maximize the discrepancy and thus adapts the source domain better towards the global domain. This can be why TCA is more effective at higher training sample sizes.

In both experiments, more training data leads to more consistent and predictable results with smaller ranges on average. This is probably because, with more training samples, classifiers better represent the underlying domains. Furthermore, the sample size of the training set hardly influences the mean

accuracies of all classifiers in the experiments. Every classifier's mean accuracy at higher sample sizes stays within a margin of *0.05* or less. SA mitigates the effects of sample selection bias better than TCA. For TCA, it seems that a higher training sample size decreases the effects of biased data on accuracy. However, this improvement is minimal.



(a) Results with Logistic Regression as an estimator for SA and TCA

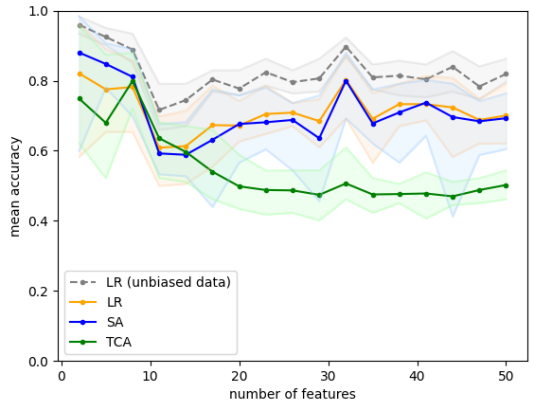


(b) Results with KNN as an estimator for SA and TCA

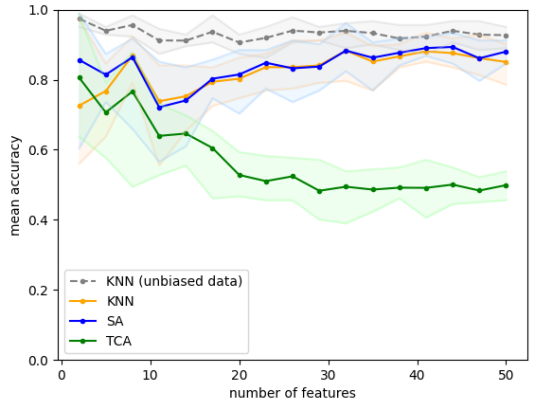
Figure 2: Mean, lowest and highest accuracies on different training sample sizes. The grey dotted line represents the mean accuracies of KNN or LR when trained on an unbiased training set

3.2 Experiment 2: Number of features

In the second set of experiments, the mean accuracies of SA and TCA were measured for different numbers of features. For this set of experiments, all data sets were generated using the *make classification* function described in Section 2.1. We set the parameters of this function to n_samples = 5000, *n_classes* = 2, *n_redundant* = 0, *n_repeated* = 0, *n_clusters_per_class* = 2, and *random_state* = None. Furthermore, *n_informative* was set to the same value as the number of features. To create a variable number of features to investigate, for every data point, a data set was generated with different numbers of features by adjusting the *n_features* parameter of the *make classification* function. An example of the form of the data set can be found in Figure 1, where *n_features* is set to 2 and *n_samples* to 500. For the biasing function, f was set to 50, and r was set to 0.1.



(a) Results with Logistic Regression as an estimator for SA and TCA



(b) Results with KNN as an estimator for SA and TCA

Figure 3: Mean, lowest and highest accuracies on different numbers of features. The Grey dotted line represents the mean accuracies of KNN or LR when trained on an unbiased training set

The results for the experiments are displayed in Figure 3. Both experiments (Figures 3a, 3b) show that SA has higher mean accuracy than TCA and LR for feature sizes 2 and 5. However, from a feature size of 8 onwards, SA performs only about as well as LR and KNN and doesn't mitigate the bias. The range for SA is also bigger than that of LR but roughly the same as KNN. From a feature size of 16 onward, TCA performs worse than LR on mean accuracy. The mean accuracy of TCA drops for larger feature sizes up until around 30 features, where it stagnates. It might be harder for the TCA method to find the accurate discrepancy between the source and global domain at larger feature sizes as the data gets more complex. This could lead to diminishing effectiveness.

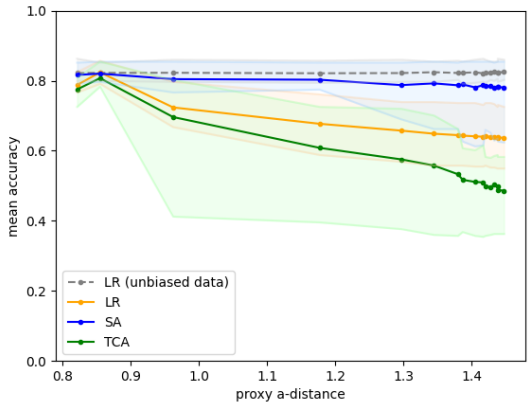
3.3 Experiment 3: Distance between source and target domain

In the third set of experiments, the mean accuracies of SA and TCA were measured against an increasing distance between the source and target domain. The distance metric used for this experiment was the proxy A-distance (Section 2.5). Our experiment used data sets generated by the *make classification* method. The parameters were set to n_samples = 10000, *n_features* = 2, *n_classes* = 2, *n_informative* = 2, *n_redundant* = 0, *n_repeated* = 0, *n_clusters_per_class* = 2, and *random_state* = 0. The data set in Figure 1 looks like the data set generated, but with *n_samples* set to 500. As the proxy A-distance gets calculated from the training (source) and test (target) data sets, we didn't have predefined proxy A-distances to use as parameters for this experiment. To simulate an increasing proxy A-distance, we increased the biasing factor f in the biasing method. The values for f we used were all integers from 0 up until 15. The biasing method's parameter r was set to 0.1.

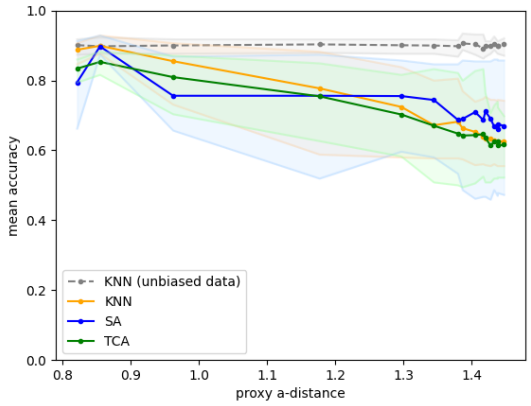
The results for the experiment can be found in Figure 4. In both experiments, the classifier trained on an unbiased training set is similar for every data point. This is because the only variable, the biasing factor, doesn't influence the unbiased training set. For LR as *estimator* (Figure 3a), SA performs better than TCA and LR. The mean accuracy of SA hardly changes for higher proxy A-distances, while the mean accuracies of LR do decrease slowly. This means that SA is slightly more effective in mitigating sample selection bias with higher proxy A-distances. Only a slight decrease in mean accuracy while the training data gets more biased suggests that SA is robust in finding the common subspace between the source and global domain. Furthermore, TCA has lower mean accuracies than LR, and this difference only increases for higher proxy A-distances. The range of TCA is also a lot higher than that of LR. This indicates that TCA struggles to adapt to the source and global domains. With KNN as *estimator* (Figure 3b), SA score better on mean accuracy than KNN at lower proxy A-distances. SA gets better mean accuracies from a proxy A-distance of 1.2 than KNN. This is mainly because the mean accuracy of SA does not change much, but the mean accuracies of KNN decrease. The range of SA, however, is bigger than that of KNN. TCA has lower mean accuracies than KNN. This difference decreases at higher proxy A-distances. The last trend in both experiments is that a higher proxy A-distance results in lower mean accuracy for all classifiers training on biased data. However, the rate at which SA's mean accuracy decreases is smaller than other classifiers and therefore gets more effective in mitigating sample selection bias with a bigger distance between the source and target domains.

3.4 Experiment 4: Different data sets

In the last experiment, the mean accuracies of SA and TCA were measured against 3 different data sets. The methods to create all of these data sets are discussed in Section 2.1. The first data set was created using the *make classification* method. The parameters for this method were set to n_samples = 10000, *n_features* = 2, *n_classes* = 2, *n_informative* = 2, *n_redundant* = 0, *n_repeated* = 0,



(a) Results with Logistic Regression as an estimator for SA and TCA



(b) Results with KNN as an estimator for SA and TCA

Figure 4: Mean, lowest and highest accuracies on different proxy A-distances. The grey dotted line represents the mean accuracies of KNN or LR when trained on an unbiased training set

n_clusters_per_class = 2, and *random_state* = 0. Figure 1 show the form that the data set takes but with *n_samples* set to 500. The second data set was generated using the *make blobs* method. The parameters for this method were set to *n_samples* = 10000, *n_features* = 2, *centers* = 12, and *random_state* = 0. An example of what the data set looks like can be found in Figure 5, but with only 500 samples. For the creation of the last data set, the *make gaussian quantiles* method was used. The parameters were set to *n_samples* = 10000, *n_features* = 2, and *random_state* = 0. Figure 6 shows what the data set looks like only with *n_samples* set at 500. For the bias technique, the bias factor f was set at 50, and the ratio of selected samples r was set at 0.1.

Table 1 displays the experiment results using LR as an estimator for SA and TCA on different data sets. SA obtained a better mean accuracy than SA than LR for the *make classification* data set, while TCA got a lower mean accuracy than LR. The results for the *make blobs* and *make gaussian quantiles* data sets show that LR trained on biased data scores on

Data set generation method		LR (unbiased data)	LR	SA	TCA
make classification	Mean	0.825	0.634	0.781	0.484
	Highest	0.854	0.717	0.854	0.582
	Lowest	0.799	0.544	0.635	0.360
make blobs	Mean	0.501	0.596	0.529	0.580
	Highest	0.574	0.637	0.593	0.648
	Lowest	0.464	0.544	0.448	0.503
make gaussian quantiles	Mean	0.482	0.594	0.465	0.748
	Highest	0.679	0.684	0.495	0.786
	Lowest	0.360	0.541	0.434	0.712

Table 1: Mean, lowest and highest accuracies of LR trained on unbiased data, LR trained on biased data, SA, and TCA on different data sets

Data set generation method		KNN (unbiased data)	KNN	SA	TCA
make classification	Mean	0.898	0.595	0.688	0.606
	Highest	0.909	0.662	0.852	0.684
	Lowest	0.885	0.541	0.500	0.516
make blobs	Mean	0.938	0.588	0.545	0.587
	Highest	0.964	0.654	0.635	0.643
	Lowest	0.912	0.503	0.489	0.522
make gaussian quantiles	Mean	0.974	0.603	0.485	0.769
	Highest	0.981	0.714	0.679	0.794
	Lowest	0.964	0.524	0.429	0.720

Table 2: Mean, lowest and highest accuracies of KNN trained on unbiased data, KNN trained on biased data, SA, and TCA on different data sets

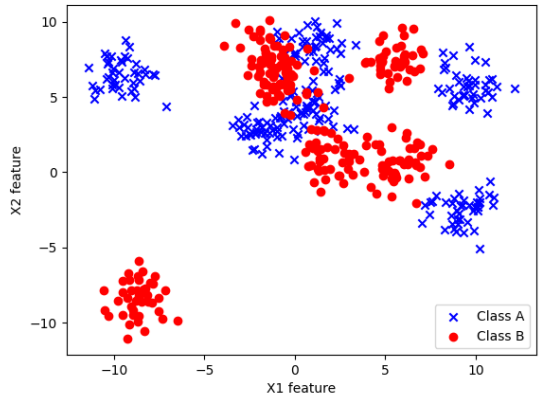


Figure 5: Data set generated by the *make blobs* function. Parameters are set to n_samples = 500, *n_features* = 2, *center* = 12, and *random_state* = 0. The rest of the parameters were set to their default values

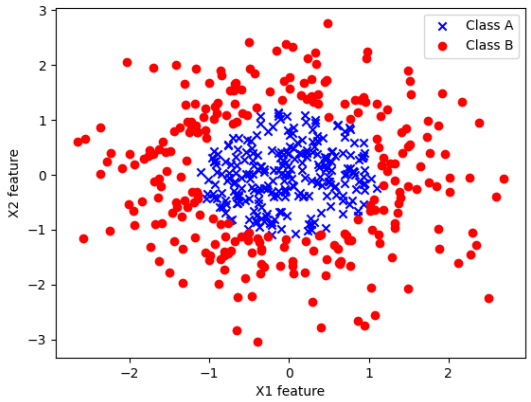


Figure 6: Data set generated by the *make gaussian quantiles* function. Parameters are set to n_samples = 500, *n_features* = 2, and *random_state* = 0. The rest of the parameters were set to their default values

average higher than LR trained on unbiased data. The reason for this could be because LR is a linear classifier, while the two data sets cannot be solved linearly. This might cause unpredictable and bad results. It is also hard to conclude whether SA and TCA mitigate the bias for these data sets, as the bias does not hurt the accuracy. Table 2 shows the results for the experiment on different data sets using KNN for the estimator parameter of SA and TCA. SA has better mean accuracy than KNN by almost 0.1 and does mitigate the bias induced for the *make classification* data set. However, the difference between the lowest and highest scores for SA is far bigger than that of KNN, and the lowest score is also lower than that of KNN. TCA also performs better than KNN, but only by 0.01 on average. On the *make blobs* data set, TCA only performs as well as KNN, while SA performs even worse. Therefore, both algorithms do not give better results on biased data. For the *make gaussian quantiles* data set, the mean accuracy of SA is much worse than that of KNN. TCA, on the other hand, has a better mean accuracy of about 0.17. So for this data set, TCA effectively reduces the bias.

Overall, the kind of data set seems to make a big difference in the effectiveness of SA and TCA. SA seems to work better for the *make classification* data set, while TCA performs better on the *make classification* data set. Both methods do not reduce the bias for the *make blobs* data set. The range of the results is also affected by the type of data set. This confirms that there is not a single solution for every data set.

4 Responsible Research

This chapter will reflect on the ethical aspects of our research and discuss how we ensured our method's reproducibility. The chapter will analyze the ethical aspects in Section 4.1. In Section 4.2, the reproducibility of the methods will be discussed.

4.1 Ethical aspects

This paper researches methods that should mitigate sample selection bias and should therefore give an objective view of the effectiveness of the methods. Failing to do so can lead to an untrue belief that the methods could reduce bias in classification problems when they might not. This could result in equally or more biased results for people who use these methods to reduce the bias in the results. Therefore, it's important that all results are correct, that conclusions drawn from the results are objective, and that left-out results should be mentioned. In this paper, we made sure all of the above apply.

4.2 Reproducibility of the methods

It is important for the research in a paper to be reproducible. An expert can validate reproducible research and is, for that reason, more trustworthy. Every implementation part should be provided or well described to ensure that all experiments are reproducible. Furthermore, all parameters used in the research should be introduced, and their configurations should be given. In our paper, we tried to add as much detail about the implementation as possible in the methodology (chapter 2). We described the data generation process, the splitting process of the data, the biasing method, the implementation of all classifiers used, the distance metric used for one of the experiments, and the evaluation criteria. All of these submethods describe a full implementation of our experiments. We also introduced every parameter used in the research in the methodology chapter. The configurations of those parameters are completely described for every experiment in chapter 3. In our implementation, there is still some randomness when generating a biased data point for our biasing technique. This results in a different data point every time the biasing technique is applied, which can lead to slightly different results with the same configuration of all parameters. To ensure this randomness doesn't impact the results too severely, we calculated each data point in every experiment as the mean of 10 iterations using the same parameters.

5 Conclusions and Future Work

This paper presents experiments done with the subspace mapping techniques SA and TCA. The classifiers map the source domain towards an underlying global domain, assuming no samples from the target domain are available. The main question in this paper was whether subspace mapping techniques adapted to the global domain effectively mitigate sample selection bias. Four sets of experiments were performed to answer this question. First of all, the influence of the training sample size on the accuracy of SA and TCA was measured. Our results indicated that a higher training sample size resulted in more consistent accuracies with lower ranges for both SA and TCA. Furthermore, the number of training samples hardly influenced the effectiveness of the classifiers. TCA was a bit more effective with more training samples. In the second set of experiments, we tested the influence of the number of features in the data sets on the accuracy of SA and TCA. For SA, more features did not improve the method's effectiveness. TCA performed worse on higher dimensional data. In the third set of experiments, the mean accuracy of SA and TCA was measured against the proxy A-distance between the source and target domains. For SA, a higher proxy A-distance improved the effectiveness of mitigating the bias by a small margin. TCA, on the other hand, performed worse at higher proxy A-distances. The last set of experiments tested all classifiers on different data sets. The main conclusion drawn from the results of this experiment is that the type of the data set impacts the effectiveness of subspace mapping techniques. From this experiment, we could also conclude that KNN as estimator for SA and TCA works better on data sets that are not linearly solvable. With *LR* as *estimator*, the mean accuracy was lower than with KNN and had a higher range on those data sets.

Overall, SA effectively mitigates sample selection bias on data sets with low features and with a high distance between the source and target domain. SA is not effective at larger feature sizes and for some specific data sets like the *make blobs* and *make gaussian quantiles* data sets. TCA is more effective with more training samples and on data sets with only a few features where the distance between the source and target domain is not too big. Since that most real-world data sets used for solving classification problems do have more than just a few features and bias should also be mitigated for domains that are further apart, TCA is not an effective method to mitigate sample selection bias.

The data sets, the *estimator* parameter, and the subspace mapping methods limit the extensiveness of all experiments in this paper. This also means that the conclusions drawn from the experiments are definite to the data sets, estimators, and subspace mapping techniques used.

To extend this paper, the effect of other subspace mapping techniques other than SA and TCA could be researched in future works. Furthermore, the methods could be implemented on different types of data sets. There are lots of other data set types that are not tested in this paper. Lastly, this paper uses only LR and KNN as *estimator* parameters of the subspace mapping techniques. In future research, the methods could be implemented using other classifiers for *estimator*.

References

[1] Antoine de Mathelin, François Deheeger, Guillaume Richard, Mathilde Mougeot, and Nicolas Vayatis. Adapt:

Awesome domain adaptation python toolbox. *arXiv preprint arXiv:2107.03049*, 2021.

[2] Basura Fernando, Amaury Habrard, Marc Sebban, and Tinne Tuytelaars. Subspace alignment for domain adaptation, 2014.

[3] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[4] Wouter M. Kouw and Marco Loog. A review of domain adaptation without target labels. 2019.

[5] Sinno Jialin Pan, Ivor W. Tsang, James T. Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks P*, 2010.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[7] Bernhard Schölkopf, John Platt, and Thomas Hofmann. *Correcting Sample Selection Bias by Unlabeled Data*, pages 601–608. 2007.

[8] Hana Ajakan Pascal Germain Hugo Larochelle François Laviolette Mario Marchand Victor Lempitsky Yaroslav Ganin, Evgeniya Ustinova. Domain-adversarial training of neural networks. *Journal of Machine Learning Research 2016*, 17:1–35, 2015.

[9] Bianca Zadrozny. Learning and evaluating classifiers under sample selection bias. In *Twenty-first international conference on Machine learning - ICML '04*. ACM Press, 2004.