Domain Adaptation

Vighnesh Chenthil Kumar
1103842

1 Introduction

Traditional machine learning (ML) problems deal with a fundamental assumption, i.e., the training and test data are drawn from an independent and identical distribution (iid).
However, in the real-world, we may have to work on a task from a data-impoverished domain, which while being similar to an already existing domain, consists of data drawn from a different distribution altogether. In most cases, this leaves us with an extremely poor ML model since the training has been done on much lesser data.
So, the aim of the project basically boils down to using out-of-domain (source domain) data to train a model that can generalise well to the in-domain (target domain) data as well.
This report is divided into 3 sections, one for each of the methods under consideration.

2 Methods

The Schools datasets consist of 3 domains (types of schools): male, female and mixed gender. Each of the 3 domains consist of 7 features, of which 4 are categorical variables. However, using categorical variables (as it is) in our experiments is synonymous to telling the model to look at it as weights, which goes against the meaning the variables.
One of the techniques to avoid this problem is to use one-hot encoding to represent each of the categorical variables, and doing so resulted in 23 total features (including the label).
The data was then partitioned domain-wise, wherein 100 randomly chosen instances from each domain were assigned for validation and testing each, while the remaining was used for training.

2.1 Baseline Methods

There are 6 baseline techniques for the domain adaptation problem, which were presented and evaluated by Daume III and Marcu (2006). Each of them differ only in the training set used, while validation and testing are done on the in-domain data itself.
The SRCONLY method trains the ML model only on the out-of-domain data, while the TGTONLY method uses only the in-domain data for training. The ALL method uses the out-of-domain data and a few instances (size = 100) of in-domain data for training, in order to simulate a data-impoverished setting. The WEIGHTED model is similar, except we upsample the 100 in-domain instances to match the size of the out-of-domain data. The PRED method trains an ML model using the SRCONLY method, and then appends the predictions of the in-domain data as an extra feature to the in-domain data itself. This augmented in-domain data is now trained again using a TGTONLY method. Lastly, we have the LININT method, wherein we linearly interpolate the predictions of the SRCONLY and TGTONLY methods.
These methods were then tested on a linear regression model (statistical learning model) and a Multi-Layer Perceptron (neural network model). The reasoning behind choosing a linear regression model was to have a (supposedly) naive model to contrast with a much "smarter" learner like MLP.
The following are the mean squared error values across all 6 baselines for each of the 2 ML models:

SRCONLY	TGTONLY	ALL	WEIGHTED	PRED	LININT
97.83	132.91	**97.50**	101.36	132.91	98.29
128.92	126.75	128.54	**117.56**	126.49	128.53
105.01	106.81	104.77	101.04	106.81	**100.09**

Table 1: Mean squared error (MSE) values for Linear Regression (LR) model.

SRCONLY	TGTONLY	ALL	WEIGHTED	PRED	LININT
99.07	121.07	92.24	107.26	125.09	**91.39**
134.58	130.11	125.68	**121.05**	133.02	124.02
105.74	102.26	**101.47**	123.46	109.61	102.12

Table 2: MSE values for the Multi-Layer Perceptron (MLP) model.

As we can see, ALL, WEIGHTED and LININT methods each perform best exactly twice across the 6 (3 domains x 2 models) domains. We can conclude that mixing some amount of in-domain data with the out-of-domain data (or even their corresponding predictions) increases the model performance. However, the PRED method tends to underperform, which challenges our conclusion. A potential reason for this performance drop can be attributed to our data sampling as well. The same reason could also hold as to why a simple model like LR beats a much smarter MLP model nearly 55% of the time.

The linear regression, being a simple model, only had one tunable hyperparameter, which proved to be quite insignificant. For tuning the best hyperparameters for the MLP model, we chose 5 hyperparameters (mentioned in the table below) which proved crucial in affecting the model performance (during our initial experiments), and ran GridSearch across nearly 162 possible parameter combinations. (Note: the max_iter hyperparameter was set to 1000 for all experiments to allow all models to converge.)

After obtaining the best parameter combination for each domain (for a single method), we performed majority voting to get the final parameter combination to be used, in order to ensure generalisation. Table 3 shows the hyperparameter values used for each baseline method. (Note: The SRCONLY methods within the PRED and LININT methods, and the TGTONLY method within the LININT method use the default parameter combination.)

Hyperparameters	SRCONLY	TGTONLY	ALL	WEIGHTED	PRED
activation	relu	identity	relu	relu	relu
alpha (L2 reg)	0.0001	0.0001	0.0001	0.01	1
batch_size	128	64	64	64	64
hidden_layer_sizes	(100,)	(100,)	(100,)	(100,)	(100,)
learning_rate_init	0.001	0.001	0.001	0.001	0.001

Table 3: Hyperparamters used for the 6 baseline methods.

2.2 FEDA Method

The FEDA method was introduced by Daume III (2009), and discusses the concept of an additional *general* domain. In our problem, we now have 4 domains, 1 general domain, 2 source domains and 1 target domain. According to FEDA, all of our 22 features are replicated 4 times, once each for the general and target domains, and initialised to zeros the remaining two times. The reasoning behind choosing the LR and MLP models, and a pre-defined set of 5 hyperparameters is the same as above. In addition, we prefer the ALL technique here, due to its inclusiveness of data from both domains. Apart from the default size of in-domain training data (100), we also try variations in size, which lead to the following MSE values for each of the 2 ML models:

n=100	n=300	n=500	n=700	n=900
132.91	94.33	92.34	93.12	**91.02**
126.68	118.37	115.82	**115.62**	116.53
106.81	97.06	95.77	95.92	**95.76**

Table 4: MSE values for Linear Regression model.

n=100	n=300	n=500	n=700	n=900
106.53	93.07	93.71	92.10	**88.72**

151.19	106.99	112.32	**100.94**	105.62
101.79	94.87	93.66	**89.39**	93.55

Table 5: MSE values for the Multi-Layer Perceptron (MLP) model.

We can observe that the FEDA method does not outperform the best of the 6 baseline methods in any of the 6 (3 domains x 2 models) domains, and only manages to beat the ALL baseline once, thus concluding that the feature space augmentation has not helped boost model performance by much. Another interesting observation is that as the size of in-domain training data increases (n=700 or 900), this tends to boost the model performance as well, which points to the obvious fact that the model tends to perform better on the in-domain as the size of in-domain data itself increases. As expected, the MLP model beats the simpler LR model nearly 90% of the time. Table 6 shows the variation in hyperparameter settings across in-domain data sizes for the FEDA method.

Hyperparameters	**n=100**	**n=300**	**n=500**	**n=700**	**n=900**
activation	relu	relu	relu	relu	relu
alpha (L2 reg)	1	1	1	0.0001	0.0001
batch_size	200	200	200	128	200
hidden_layer_sizes	(100,)	(50,50)	(100,)	(30,30,30)	(100,)
learning_rate_init	0.001	0.001	0.001	0.001	0.001

Table 6: Hyperparamters used for variations of FEDA.

2.3 Variant of TrADABOOST Method

After going through the survey of transfer learning techniques by Pan and Yang (2009), I explored a range of work including the PRIOR model (Chelba & Acero, 2006), the MEGA model (Daume III & Marcu, 2006), Structural Correspondence Learning (Blitzer et. al., 2006), the TrBagg model (Kamishima et. al., 2009) and the TrAdaBoost model (Dai et. al., 2007).

A variant of the TrAdaBoost has been adopted for our experiments. The basic idea is to automatically adjust weights of the training instances (boosting) to emphasize on the same-distribution data (in-domain data). Meanwhile, the different distribution data (out-of-domain data) is simply considered as additional training data to boost the confidence of the ML model, and its weight is automatically reduced to weaken its impact.

The algorithm is provided here, and the notations Ts and Td refer to the same-distribution data (of size m=100) and different-distribution data (of size n) respectively, while S refers to the unlabelled test-set of the same-distribution data (although it will not be used in the algorithm). Please note that the Learner refers to any ML model, and we have picked a Decision Tree model (as we wanted to observe the effects of boosting using decision trees) apart from a Linear Regression model. The h(x) and c(x) functions here refer to Learner predictions as well as the true values respectively. After the Nth iteration of the algorithm, we use the Learner from the latest iteration to perform prediction on S. The reasoning for doing so is that after N iterations (to converge), the model has been learnt by

---

Input the two labeled data sets T_d and T_s, the unlabeled data set S, a base learning algorithm **Learner**, and the maximum number of iterations N.

Initialize the initial weight vector, that $\mathbf{w}^1 = (w_1^1, \ldots, w_{n+m}^1)$. We allow the users to specify the initial values for $\mathbf{w}^1$.

For $t = 1, \ldots, N$

1. Set $\mathbf{p}^t = \mathbf{w}^t / (\sum_{i=1}^{n+m} w_i^t)$.
2. Call **Learner**, providing it the combined training set T with the distribution $\mathbf{p}^t$ over T and the unlabeled data set S. Then, get back a hypothesis $h_t : X \to Y$ (or $[0, 1]$ by confidence).
3. Calculate the error of h_t on T_s:

$$\epsilon_t = \sum_{i=n+1}^{n+m} \frac{w_i^t \cdot |h_t(x_i) - c(x_i)|}{\sum_{i=n+1}^{n+m} w_i^t}.$$

4. Set $\beta_t = \epsilon_t / (1 - \epsilon_t)$ and $\beta = 1/(1 + \sqrt{2 \ln n / N})$. Note that, ϵ_t is required to be less than $1/2$.
5. Update the new weight vector:

$$w_i^{t+1} = \begin{cases} w_i^t \beta^{|h_t(x_i) - c(x_i)|}, & 1 \le i \le n \\ w_i^t \beta_t^{-|h_t(x_i) - c(x_i)|}, & n+1 \le i \le n+m \end{cases}$$

Figure 1: TrAdaBoost Algorithm

focusing on the data-impoverished same-distribution data, and is expected to perform well in that domain. The MSE values for each of the 2 models for all 3 domains are given below:

Linear Regression	Decision Trees
101.49	114.42
118.46	147.43
101.97	121.94

Table 7: MSE values for the "inspired" TrAdaBoost method

As we can observe, the Linear Regression model using the boosting technique has outperformed the FEDA method for the same size (100) of in-domain training data, and is also comparable to some of the better baseline models like WEIGHTED. Hopefully, we will be able to beat the baseline models by using better models in the future.

The hyperparameters have been tuned using GridSearch again, and Table 8 shows the hyperparameter values used for each ML model.

Hyperparamters	Linear Regression	Decision Trees
N	3	50
criterion	-	gini
max_features	-	log2
splitter	-	random

Table 8: Hyperparamters used for each of the 2 ML models.

References

Blitzer, J., McDonald, R., & Pereira, F. (2006, July). Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 conference on empirical methods in natural language processing* (pp. 120-128).

Chelba, C., & Acero, A. (2006). Adaptation of maximum entropy capitalizer: Little data can help a lot. *Computer Speech & Language*, *20*(4), 382-399.

Dai, W., Yang, Q., Xue, G. R., & Yu, Y. (2007, June). Boosting for transfer learning. In *Proceedings of the 24th international conference on Machine learning* (pp. 193-200).

Daumé III, H. (2009). Frustratingly easy domain adaptation. *arXiv preprint arXiv:0907.1815*.

Daume III, H., & Marcu, D. (2006). Domain adaptation for statistical classifiers. *Journal of artificial Intelligence research*, *26*, 101-126.

Kamishima, T., Hamasaki, M., & Akaho, S. (2009, December). TrBagg: A simple transfer learning method and its application to personalization in collaborative tagging. In *2009 Ninth IEEE International Conference on Data Mining* (pp. 219-228). IEEE.

Pan, S. J., & Yang, Q. (2009). A survey on transfer learning. IEEE Transactions on knowledge and data engineering, 22(10), 1345-1359.