EECS 440 Final Project: Transfer Learning

David Blincoe, Brendan Dowling, Kennan LeJeune,

Sam Jenkins, Chris Toomey, Arthur Xin

December 7, 2019

Instance Transfer with Boosting

David Blincoe

1 Introduction

One approach to inductive transfer learning is to use instances from the source and target domain to determine the overlap of the underlying data distributions. One approach to this is to define the *same-distribution* as the distribution under which the target domain falls, and the *diff-distribution* as the distribution under which the source domain falls. Again, one of the primary focuses of transfer learning is to avoid labeling target distribution examples.

TrAdaBoost takes the approach of requiring some labeled target distribution data, or *same-distribution* data. The algorithms focus and goal is train on a large quantity of *diff-distribution* data that is supplemented with this small amount of *same-distribution* data. Logically, this can be thought of as an assumption that some of the *diff-distribution*'s data will fall within the *same-distribution*. A specific approach can be taken to transfer this underlying knowledge from the source to target goals. This is known as Instance-Transfer.

One easy way to facilitate the learning of which examples from the *diff-distribution* is through weighting each example and allowing those examples weights to impact whether the learning. AdaBoost is very effective at iteratively weighting the training examples and determining which examples need to be considered more heavily than others.

This works by increasing the weight of an example if the example was predicted incorrectly by the classifier. When an example is predicted correctly, it is assumed that the example is learned and the model has fit to it. The weight is then decreased and the algorithm assigns a weight to this iteration of the classifier. Once the algorithm either achieves an error of 0 or hits a maximum iteration cap, this process is suspended.

Below, TrAdaBoost is presented along with a modification to allow for multiple sources of data

2 TrAdaBoost

TrAdaBoost operates in the same fashion except is updates the weights of the same-distribution and diff-distribution differently.

2.1 Formal Definition

Before describing the inner-workings of the algorithm some formal definitions must be defined.

Let there be some sample space X_S that falls under the *same-distribution* and another sample space X_D that is under the *diff-distribution*. The label sample space is defined as $Y = \{0, 1\}$. Together, $X_S \bigcup X_D = X$ defined the entire sample space under which TrAdaBoost will operate.

The train data for the algorithm is defined as two sets, T_D and T_S. $T_D = \{(x_i^d, f(x_i^d)\}$ where $x_i^d \in X_D$ and $(i = 1, \ldots, n)$. $T_S = \{(x_i^s, f(x_i^s)\}$ where $x_i^s \in X_S$ and $(i = 1, \ldots, m)$. Here n and m are the sizes of two sets. Here $f(x)$ defines the function mapping the example from the given sample space to the output space. Together these sets form the training data. [1]

$$x_i = \begin{cases} x_i^d & i = 1, \ldots, n \\ x_i^s & i = n+1, \ldots, n+m \end{cases} \tag{1}$$

The testing data is defined as one set, S. $S = \{(x_i^t)\}$ where $x_i^t \in X_S$ and $(i = 1, \ldots, k)$. K is the size of the test set. [1]

Logically, the above definitions are as follow. T_D is the data from the *diff-distribution* that is being reused to learn this problem and T_S is from the limited amount of labeled data in *same-distribution*. The overall goal of our classifier is to learn some function $\hat{f}$ such that the training and testing error is minimized.

2.2 Theory

In the TrAdaBoost paper, [1], the authors expose some of the underlying theory of the modification to AdaBoost.

Two core theorems of AdaBoost are listed below and these define why AdaBoost will converge on a solution. These same theorems also hold with TrAdaBoost.

$$\frac{L_d}{N} \leq \min_{1 \leq i \leq n} \frac{L(x_i)}{N} + \sqrt{\frac{2 \ln n}{N}} + \frac{\ln n}{N} \tag{2}$$

With 2, this guarantees that it will converge on some value eventually.

$$\min_{N \to \infty} \frac{\sum_{t=N/2}^{N} \sum_{i=1}^{n} p_i^t l_i^t}{N - (N/2)} = 0 \tag{3}$$

In 3, this shows that the weighted training error will eventually converge to zero.

Through the above two equations, this will eventually find some best fit classifier for some given problem. Even though TrAdaBoost is proven theoretically to work, it cannot be guaranteed that it will always perform better than AdaBoost [1].

2.3 Algorithm

This modification to AdaBoost follows the same procedure as in normal AdaBoost, except for the weighting of the T_D data. As in AdaBoost, when an example from the T_S set is predicted correctly, it is assumed learned and its weight is decreased and when it is predicted incorrectly, the examples weight is increased. For the T_D data, when the model predicts an example incorrectly, an assumption is made that this example lies outside of the *same-distribution* and the example weight is decreased. Likewise, the inverse applies.

Algorithm 1 `TrAdaBoost`

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:

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

Output the hypothesis

$$h_f(x) = \begin{cases} 1, & \prod_{t=\lceil N/2 \rceil}^{N} \beta_t^{-h_t(x)} \ge \prod_{t=\lceil N/2 \rceil}^{N} \beta_t^{-\frac{1}{2}} \\ 0, & \text{otherwise} \end{cases}$$

Figure 1: TrAdaBoost Algorithm [1]

2.4 Analysis of Single Source Domain

For all measures below, the given settings of AdaBoost are 20 iterations, 10% of the target domain used in training, and $10,000$ features in the bag of words feature vector.

2.4.1 Amazon Review Data Analysis

Determining Best Matching (-i 20 -p 0.1 -f 10000)				
Distribution to Distribution	Accuracy	Precision	Recall	AROC
Books to Kitchen	0.821, 0.016	0.887 0.052	0.741 0.07	0.894
Books to DVD	0.812, 0.008	0.84, 0.012	0.773, 0.011	0.893
Books to Electronics	0.794, 0.006	0.878, 0.026	0.687, 0.041	0.884
DVD to Books	0.763, 0.151	0.766, 0.154	0.866, 0.084	0.892
DVD to Electronics	0.756, 0.129	0.805, 0.157	0.783, 0.123	0.894
DVD to Kitchen	0.772, 0.154	0.804, 0.18	.832, 0.107	0.902
Electronics to Books	0.717, 0.129	0.71, 0.128	0.845, 0.101	0.844
Electronics to DVD	0.796, 0.027	0.778, 0.049	0.839, 0.034	0.861
Electronics to Kitchen	0.864, 0.015	0.83, 0.039	0.915, 0.017	0.934
Kitchen to Books	0.794, 0.006	0.791, 0.045	0.812, 0.57	0.858
Kitchen to DVD	0.807, 0.02	0.8, 0.038	0.826, 0.026	0.876
Kitchen to Electronics	0.793, 0.163	0.824, 0.18	0.855, 0.081	0.931

Figure 2: Best Amazon Domain Transfer

In 2, the highest performing transfer was Electronics to Kitchen with an AROC of 0.934 and an accuracy of 0.864. The second best transfer was Kitchen to Electronics with an AROC of 0.931 and accuracy of 0.793. Both the first and best transfers are inverses of one another which may be indicative of some link underlying link.

Source Domain	Average Accuracy
Books	0.809
DVD	0.764
Electronics	0.792
Kitchen	0.798

Figure 3: Amazon Domain Accuracy Average

If the accuracy metric is averaged over the source domain, 3 is achieved. This shows that overall the best source domain is books while the worst is DVD.

Figure 4: Amazon Domain AROC of TrAdaboost vs AdaBoost

In 4, the AROC of TrAdaBoost is compared to the AROC of AdaBoost. The amount of target domain data is varied and the testing AROC is outputted. Here we can see TrAdaBoost performing better than AdaBoost for whenever *same-distribution* data is given.

2.4.2 Task A Spam Data Analysis

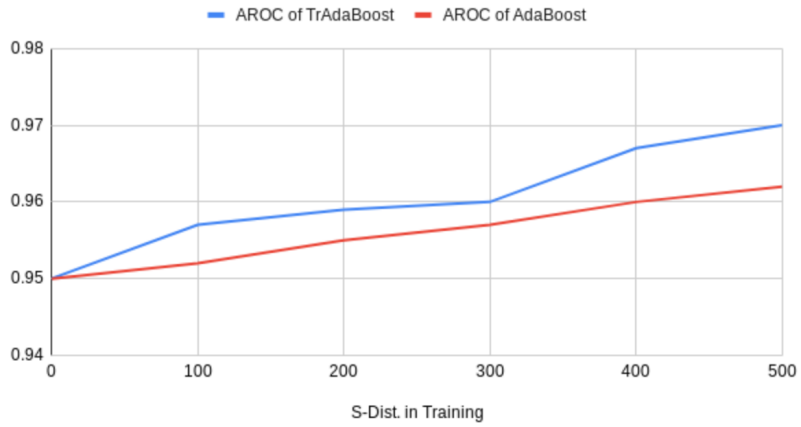


Figure 5: Task A Domain AROC of TrAdaboost vs AdaBoost

In 5, the same comparison is made as in 4. TrAdaBoost routinely performs better than AdaBoost if both algorithms are given the same training data and parameters.

2.4.3 Newsgroups Data Analysis

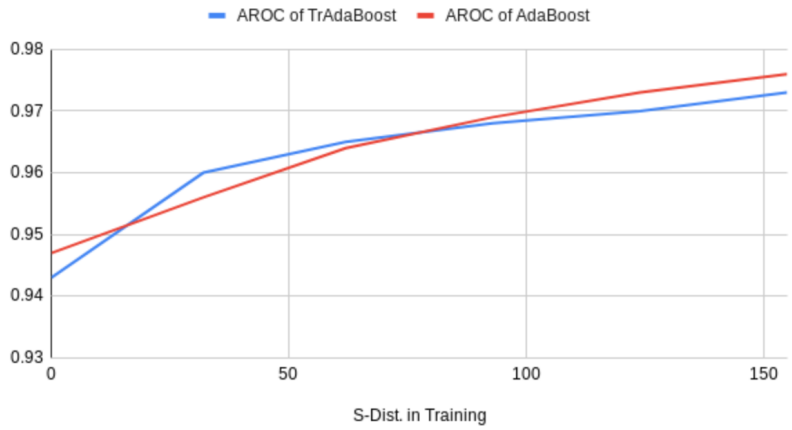


Figure 6: Newsgroups Domain AROC of TrAdaboost vs AdaBoost

Again in figure 6, target data is slowly increased and the AROC of TrAdaBoost and and AdaBoost are compared. Unlike in 4 and 5, TrAdaBoost did not outperform AdaBoost.

2.4.4 Single Domain Comparisons

In the above sections (2.4.1, 2.4.2, 2.4.3) comparisons of TrAdaBoost versus its baseline, AdaBoost are made.

For both Amazon and Task A Spam, the transfer learning approach is always better than no transfer learning. For the newsgroup data set, using AdaBoost yielded a higher AROC score and when compared in a t-test, AdaBoost was statistically better than TrAdaBoost. In 2.2 there is no guarantee that transfer learning in this context will improve prediction scores.

3 Multisource TrAdaBoost

One extension to TrAdaBoost is allowing the algorithm to train on multiple source domains to learn a target domain.

3.1 Formal Definition

This definition follows closely with 2.1 were T_D and T_S were defined to compose the training set and S was defined as the set of examples to test on from the *same-distribution*. When converting to Multisource TrAdaBoost, T_S and S remain the same for some given target domain.

Here $T_D = \{(x_1^d, f(x_1^d), \ldots, (x_{n_s}^d, f(x_{n_s}^d)))\}$ where x_i^d is a set of points chosen from the i^{th} source domain [2].

3.2 Algorithm

The algorithm operates closely to AdaBoost except for step 2 of 1. Instead of simply creating a weak learner based on a single source domain, we now iterate over the multiple source domains and determine which domain yields the lowest weighted error. Let that weak classifier be added to the list of weak classifiers that compose AdaBoost and let the given error $= \epsilon_i$ [2].

3.3 Analysis of Multisource Domain

In the following analysis, the Amazon Review data set and Task B Spam data are run comparing how the multisource version of TrAdaBoost compares to single source domain transfer.

3.3.1 Amazon Review Data Analysis

To evaluate the Amazon Review data over multiple sources, I select one target domain and use the remaining 3 domains to perform training. In the following figure, 7, it can be seen that the Kitchen domain reacts the best to using multiple sources and books responds the least.

Target Distribution	Accuracy	Precision	Recall	AROC
Kitchen	0.842, 0.012	0.867, 0.034	0.874, 0.041	0.923
DVD	0.801, 0.03	0.85, 0.21	0.864, 0.052	0.914
Electronics	0.811, 0.20	0.862, 0.26	0.840, 0.023	0.937
Books	0.799, 0.31	0.821, 0.023	0.834, 0.053	0.892

Figure 7: Multisource Amazon Reviews

Comparing the accuracies of the multisource problem to that of 3, every accuracy is higher in the multisource setup. It is very probable that including every domain in the classification problem helped push the accuracies higher. This is because the classifier is better able to generalize itself to the problem of sentiment analysis of reviews, independent of review domain.

3.3.2 Task B Spam Data Analysis

To evaluate multisource performance on the Task B Spam data, the problem was run by using the first 2 domains as the source and the third domain as the target.

Accuracy	Precision	Recall	AROC
0.862, 0.023	0.885, 0.041	0.902, 0.027	0.921

Figure 8: Multisource Task B Spam

Figure 8 is the output from the multisource problem over Task B Spam. While it is not the same problem, the classifier performed roughly equal to the classifier of Task A in 2.4.2.

4 Conclusions

The Amazon Review data showed that, based on the combined accuracy, precision, and recall scores, Kitchen is the easiest target domain to predict. This may be because kitchen reviews use the most words to predict its overall sentiment. Transferring from books yielded the highest results for when it was placed in the source group possibly because its larger size.

Within the single source problem 2, Amazon and Task A spam, TrAdaBoost outperformed vanilla AdaBoost. For Newsgroups AdaBoost outperformed TrAdaBoost. This is most likely because the source and target distributions over Newsgroups were too similar and that led AdaBoost to perform better with its simpler approach.

When applying the multisource solution to the Amazon Review data, every target domains classification metrics increased over single domain classification. With a larger, more diverse vocabulary to train on multisource classification in this setting allowed for a general knowledge of reviews.

Multisource classification did not make a significant difference when applying it to the problem of spam classification on Task B.

References

[1] W. Dai, Q. Yang, G.-R. Xue, and Y. Yu, "Boosting for transfer learning," *Proceedings of the 24th international conference on Machine learning - ICML 07*, 2007. DOI: 10.1145/1273496.1273521.

[2] Y. Yao and G. Doretto, "Boosting for transfer learning with multiple sources," *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010. DOI: 10.1109/cvpr.2010.5539857.

Frustratingly Easy Domain Adaptation

Brendan Dowling

Algorithm

Frustratingly Easy Domain Adaptation (FEDA), also known as the Feature Augmentation Method (FAM) and EasyAdapt (EA), so named because of how frustratingly well it performs compared to its complexity, is an algorithm created by Hal Daumé III that allows for better learning of a target domain by augmenting the feature space in a simple way and using the augmented feature space as the learning problem for a normal underlying learning algorithm [1]. This approach has the dual benefits of being simple to implement but also ensuring that the new learning problem will be learned no worse than the original problem.

FEDA is an algorithm for Inductive Transfer Learning in a fully supervised environment, with access to a large labeled source domain and access to a significantly smaller labeled target domain, where the source and target domains are reasonably similar. This is specifically where we have some domains D_S, D_T being the source and target domains respectively with the same feature space X being some $\mathbb{R}^F$ with F being the number of features and some subsets d_s, d_t which are the subset of examples that we have from each domain where the size of d_s is likely significantly larger than the size of d_t. Our goal is then to establish a function $f : X \rightarrow Y$ with X again being an input space and Y being a label.

FEDA functions by augmenting the feature space X by, in the case with one source domain and one target domain, and transforming the feature space from $X = \mathbb{R}^F$ to $X' = \mathbb{R}^{3F}$ where we augment the feature space with disjoint additional representations of the original features along with a zero matrix. Specifically we define a mapping from X to X' for the source and target feature spaces as:

$$\Phi_s(x) = (x, 0, x) \ , \ \Phi_t(x) = (x, x, 0)$$

where **0** is a vector of F zeroes.

It is important to consider why in fact this makes sense as a mapping that will allow for a better learning of the problem. Consider the example of the Multi-Domain Sentiment Dataset where we deal with sentiment classification. More specifically consider only the source domain being “kitchen” and the target domain being “electronics” where each domain only has the subset of words “hot” and “nice”. We would create our new feature space as $(x_1, x_2, x_3, x_4, x_5, x_6)$ with x_1, x_2 corresponding to “hot” and “nice” in both domains, x_3, x_4 corresponding to “hot”

and “nice” in the source domain, and x_5, x_6 corresponding to “hot” and “nice” in the target domain.

In this example we may expect that the word “nice” has an association with a positive sentiment in both domains and we may then expect our feature space to assign some positive weighting $(0, 1, 0, 0, 0, 0)$. This means that in the portion of the augmented feature space that corresponds to both classifiers we are assigning a positive value to that feature, correlating it with positive sentiment for both cases. However, hot for kitchen may correspond for a positive sentiment, such as people giving positive reviews about kitchen heating items, whereas for electronics it may correspond to a negative sentiment, possibly being a negative response to electronics overheating. In this case the learning algorithm has the ability to independently relate the features to sentiments for each of the domains, creating a weighting of $(0, 0, -1, 0, 1, 0)$ relating to how the word corresponds to a positive sentiment only in the target domain based on our limited number of examples, whereas it corresponds with a negative domain in the context of the source examples. This means that our augmented feature space allows the learning algorithm to learn generalized concepts that allow for differences to exist between the relation between the words to sentiments for each domain, but use the source to learn the general idea of what is a review with a positive sentiment. This accounts for a problem known as context feature bias [2] which occurs when the same feature in different contexts is associated with a different classification.

It is also simple to generalize this beyond a single source domain and use this for multiple domains. If we were to use n source domains then we can change our feature space from $X' = \mathbb{R}^{3F}$ to $X' = \mathbb{R}^{(n+2)F}$ where we would have n for each of the source domains, one set of features for the target domain, and another for the combination of domains.

The primary positive feature of this algorithm is that in theory it should never lead to a worse learned classifier than just learning on the target data, no matter what the data is. This is explained as being due to optimizing the weights with respect to the source and target domains jointly. This allows a single learning algorithm to self-regulate the tradeoff between the individual weights of the separate source and target domains with the wights of the generalized features, which all domains would use.

In addition to the basic algorithm, Daumé III’s paper also includes in the discussion a mention to a further task being using a similarity metric to tune how the data is trained on the source domains. I made this modification by including a Jaccard similarity metric which is simply the size of the intersection of the domains over the size of the union of the domains

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

This makes sense as a very simple similarity metric as if the features for two domains are entirely the same set we see that the domains have a similarity of 1, but if there is no intersection between the domains then the Jaccard similarity is 0. This allows for a very quick mechanism to reduce the weight of the examples in the source domains respective to their similarity to the target domain. There are both benefits and issues with this approach as the similarity metric is only taking the actual features and comparing those, even if the features would in fact have different connotations in each domain. However, the assumption is that if the domains are highly similar in the language that they use it is also likely that they are similar in how the words relate to the classification.

Implementation

FEDA, true to its name, is not a complex algorithm to implement. As it operates in such a way to create an easier problem for standard learning algorithms to learn. As such I used the scikit learn library for underlying learning algorithms as FEDA is entirely algorithm agnostic, as well as scikit learn's dictvectorizer to quickly transform my features from a dictionary to a matrix. Most of FEDA is relatively simple to implement but there were a few deviations from the obvious that I included within my program. The first and most important deviation is that I cut down the features included from the full set of features to the N most frequent features. This does run the risk of removing uncommon but entirely predictive words from the vocabulary used but testing with the full set of data indicated that the accuracy does not change significantly with a cut down feature set. This also allows for another axis of comparison for the data by varying the size of the feature space used. Additionally because we are agnostic to the underlying learning algorithm we are not limited to the algorithm implemented in the source paper and can instead use a different algorithm which gives better performance. I also implemented the previously described modification using Jaccard Similarity in order to

In addition to implementing FEDA I also took the opportunity to implement three baselines to compare my algorithm to, being the three baseline algorithms referenced in Daumé III's paper. These are the *source-only* baseline which trains a classifier on exclusively the source domain, the *target-only* baseline which trains a classifier exclusively on the target domain, and the *all* baseline which trains a classifier on the union of two domains. All of these classifiers are then evaluated on the target domain.

Analysis

First in order to show the difference in the underlying algorithm between having an underlying logistic regression classifier compared to an underlying SVM as is in the paper and give justification for using logistic regression for the rest of the analysis first we show how the Multi-Domain Sentiment Dataset performs with the different underlying algorithms with 100 target domain examples over 20 iterations using the top 100

Accuracy±Deviation	SVM	Logistic Regression
Source	0.7588±0.0026	0.7752±0.0021
All	0.7832±0.0081	0.7933±0.0066
Target	0.7384±0.0193	0.7430±0.01796
FEDA	0.7938±0.01254	0.8019±0.0110

Table 1

As can be seen from Table 1 every baseline and algorithm performs better using underlying logistic regression rather than the SVM implementation of FEDA used in the source paper. We find that using underlying Logistic Regression is both more accurate and has a lower variance when compared to SVM. Thus the remainder of this paper will only be considering the underlying learner being logistic regression.

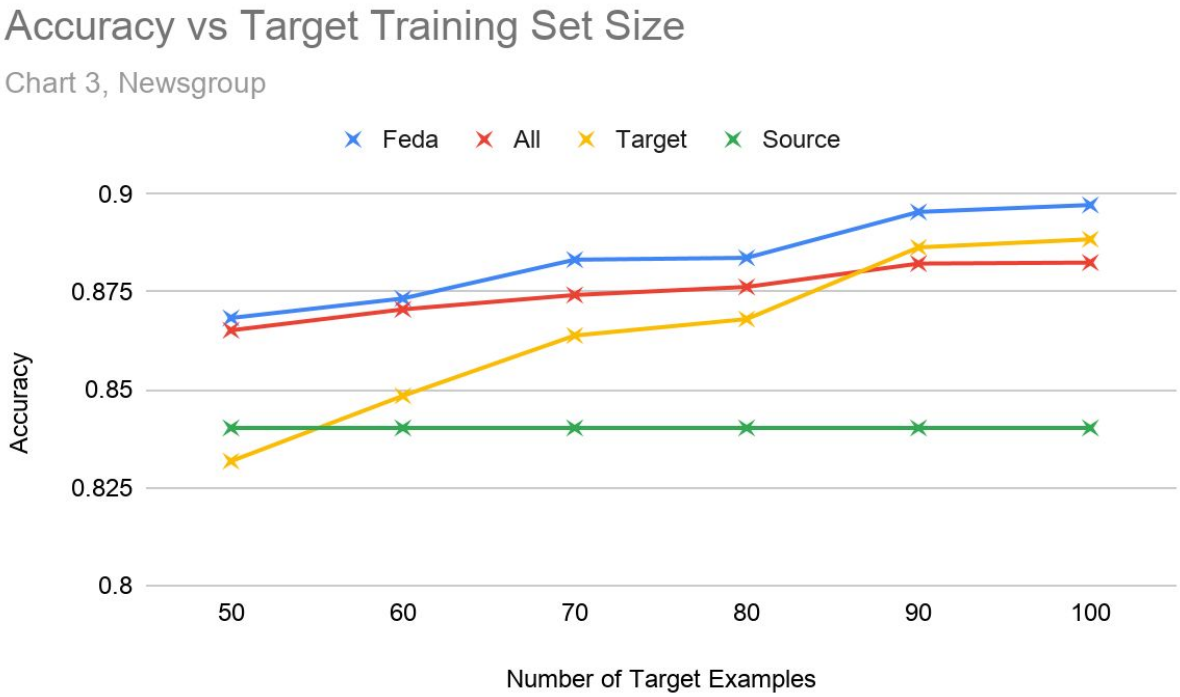
The first major axis of comparison for arbitrary datasets was understanding how much impact the number of labeled target examples we are using has on the accuracy of FEDA. In this case data was taken using the Multi-Domain Sentiment Dataset. using the three baselines of training the underlying learning model with just the target data, just the source data and the union of both. The average accuracy over 20 iterations of the algorithm was taken along with the standard deviation which is included in the appendix data. The feature space was cut down to 10000 of the highest frequency features.

The *source* baseline as we can see did not vary at all due to the size of its dataset being entirely consistent. Both the *target* baseline and the *all* baseline increased with respect to the size of the target example dataset included within the training data, as was expected. FEDA was consistently more accurate than all of the baselines and only increased in accuracy compared to them with respect to the size of the available data.

For the spam classification problem, **ECML/PKDD task a,** using 10000 features and 20 iterations we found the following data by varying the size of the target examples given.

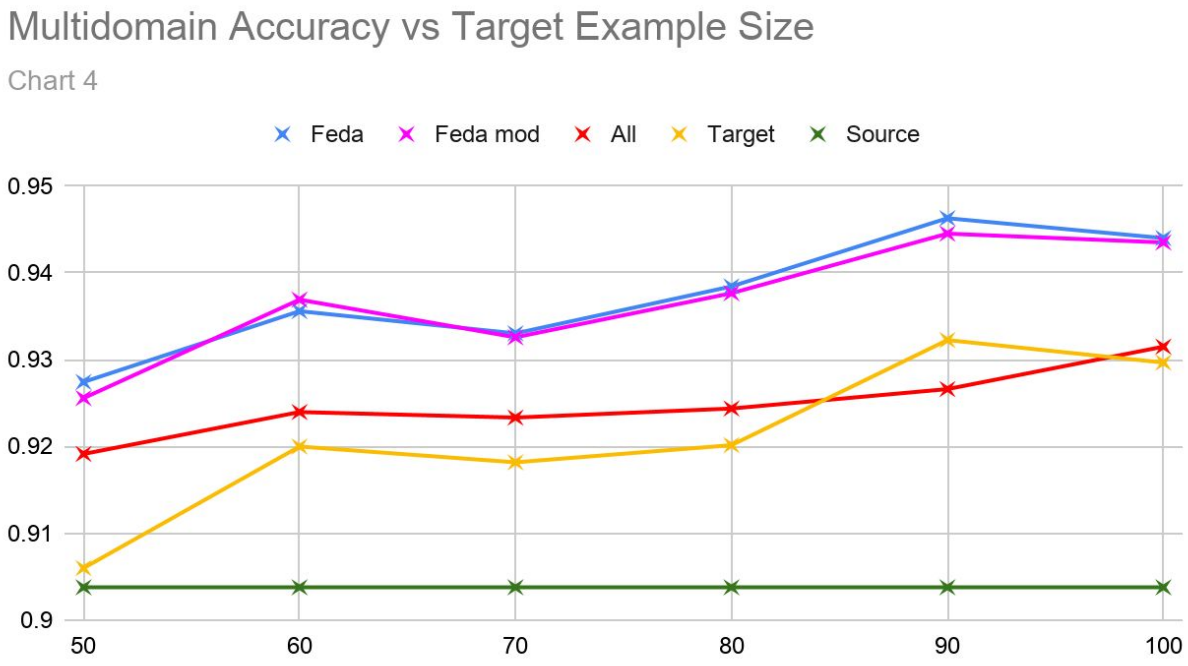
In this dataset FEDA consistently underperformed compared to both the *all* baseline and the *source* baseline.

For the newsgroup classification problem using the **20 Newsgroups** dataset using 10000 features and 20 iterations we found the following data by varying the size of the target examples given.



In this dataset we saw that there was consistent accuracy improvement of FEDA over all of the baselines.

In the multidomain **task b** spam classification problem using two source domains with the 10000 most frequent features and 20 iterations we found the following data



As we can see from the data we have a consistent improvement in accuracy using FEDA but the modified version of FEDA, while having consistent improvement over the baseline, does not have a consistent improvement over the regular version of FEDA.

Conclusions

According to the t tests calculated for each of the datasets for 100 target labeled examples we find that with 95% confidence FEDA performs better than the highest baseline, being *all* in each case, on all datasets except for **ECML/PKDD task a**. This is in line with the existing literature, both in how it outperforms the data and how it underperforms. As noted in Weiss Et. Al. [2] FEDA does underperform on datasets if the target and source domains are similar, due to the duplication of features leading to irrelevant noise rather than allowing for the smaller overlap to inform the classification of the target examples. Otherwise however, FEDA consistently outperforms all other baselines, showing that a small advantage for transfer learning can consistently cause a 1-2% increase in accuracy. The modified implementation of FEDA for the multidomain case did not show a statistically significant increase in accuracy, or any increase in accuracy at all. This could be because of the particular dataset it was applied in or because Jaccard Similarity is not a useful similarity metric due to the fact that contextually the features may be correlated with significantly different classes.

Further Topics

Daumé III created an extenstion to FEDA in Easy Adapt++ [3] which instead of only operating on the fully supervised domain also operates on the semi-supervised domain, where only the source domain is labeled. This addresses one of FEDA's most significant shortcomings as it requires some portion of the target examples to be labeled in order to be used. Easy Adapt++ fundamentally functions by having the source domain hypothesis and the target domain hypothesis "agree" on the appropriate label for the unlabeled data. This would be an appropriate further direction for work in this area as it is an augmentation of FEDA.

Bibliography

[1]H. Daume III, "Frustratingly Easy Domain Adaptation", *ACL 2007*, 2007. Available: https://arxiv.org/abs/0907.1815. [Accessed 8 December 2019].

[2]K. Weiss, T. Khoshgoftaar and D. Wang, "A survey of transfer learning", *Journal of Big Data*, vol. 3, no. 1, 2016. Available: 10.1186/s40537-016-0043-6 [Accessed 8 December 2019].

[3]H. Daume III, "Frustratingly easy semi-supervised domain adaptation", *DANLP 2010 Proceedings of the 2010 Workshop on Domain Adaptation for Natural Language Processing*, pp. 53-59, 2010. Available: https://dl.acm.org/citation.cfm?id=1870534. [Accessed 8 December 2019].

Appendix

Full Data for Charts

Chart 1 Data

Alg/Tgt	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190
Feda	0.7818	0.7857	0.7870	0.7914	0.7932	0.7999	0.8039	0.8035	0.8073	0.8056	0.8092	0.8125	0.8163	0.8165	0.8152	0.8198
All	0.7830	0.7852	0.7883	0.7886	0.7917	0.7936	0.7979	0.7975	0.7989	0.8001	0.8015	0.8055	0.8061	0.8075	0.8099	0.8082
Target	0.6785	0.6906	0.7096	0.7228	0.7292	0.7420	0.7515	0.7486	0.7586	0.7563	0.7653	0.7682	0.7738	0.7765	0.7785	0.7883
Source	0.7742	0.7748	0.7739	0.7740	0.7745	0.7743	0.7743	0.7745	0.7741	0.7748	0.7747	0.7745	0.7757	0.7757	0.7751	0.7749
Feda std dv	0.0168	0.0160	0.0128	0.0126	0.0089	0.0145	0.0103	0.0111	0.0090	0.0096	0.0104	0.0114	0.0097	0.0070	0.0081	0.0078
All std dv	0.0046	0.0049	0.0062	0.0060	0.0067	0.0058	0.0057	0.0063	0.0068	0.0075	0.0066	0.0072	0.0070	0.0050	0.0082	0.0076
Target std dv	0.0388	0.0269	0.0218	0.0232	0.0224	0.0210	0.0161	0.0204	0.0119	0.0155	0.0138	0.0186	0.0212	0.0134	0.0089	0.0091
Source std dv	0.0013	0.0012	0.0014	0.0019	0.0019	0.0013	0.0016	0.0018	0.0018	0.0023	0.0027	0.0030	0.0033	0.0027	0.0030	0.0034

Chart 2 Data

Alg/Tgt	50	60	70	80	90	100
Feda	0.9352	0.9363	0.9394	0.9405	0.9450	0.9459
All	0.9518	0.9552	0.9547	0.9567	0.9568	0.9591
Target	0.8599	0.8746	0.8876	0.8938	0.9009	0.9081
Source	0.9449	0.9453	0.9450	0.9454	0.9453	0.9452
Feda std dv	0.0116	0.0133	0.0125	0.0123	0.0119	0.0099
All std dv	0.0035	0.0057	0.0050	0.0034	0.0057	0.0039
Target std dv	0.0301	0.0198	0.0203	0.0213	0.0191	0.0190
Source std dv	0.0005	0.0007	0.0007	0.0008	0.0008	0.0008

Chart 3 Data

Alg/Tgt	50	60	70	80	90	100
Feda	0.8684	0.8733	0.8832	0.8837	0.8954	0.8972
All	0.8652	0.8706	0.8742	0.8763	0.8822	0.8825
Target	0.8318	0.8485	0.8639	0.8681	0.8864	0.8885
Source	0.8403	0.8403	0.8403	0.8403	0.8403	0.8403
Feda st	0.0141	0.0164	0.0148	0.0102	0.0089	0.0120
All st	0.0092	0.0080	0.0070	0.0066	0.0077	0.0089
Target st	0.0406	0.0275	0.0346	0.0245	0.0196	0.0193
Source st	0.0038	0.0038	0.0038	0.0038	0.0038	0.0038

Chart 4 Data

Alg/Tgt	50	60	70	80	90	100
Feda	0.9274	0.9356	0.9330	0.9384	0.9463	0.9440
Feda mod	0.9256	0.9369	0.9326	0.9377	0.9445	0.9435
All	0.9191	0.9240	0.9233	0.9244	0.9266	0.9315
Target	0.9060	0.9200	0.9182	0.9202	0.9323	0.9297
Source	0.9041	0.9050	0.9038	0.9009	0.9027	0.9058
Feda std dv	0.0118	0.0196	0.0175	0.0152	0.0152	0.0198
Feda Mod std dv	0.0109	0.0197	0.0188	0.0175	0.0155	0.0187
All std dv	0.0074	0.0105	0.0097	0.0126	0.0133	0.0146
Target std dv	0.0174	0.0211	0.0208	0.0211	0.0206	0.0207
Source std dv	0.0046	0.0050	0.0065	0.0081	0.0079	0.0086

T-Tests

Sentiment: 95% confidence interval 0.0006788301120 to 0.0113211698880

Single domain spam task a: 95% confidence interval -0.01806773927958 to -0.00839059392042

20 Newsgroup: 95% confidence interval 0.00793742902452 to 0.02147433577548

Multi-domain spam task b: 95% confidence interval 0.0013701779377 to 0.0236298220623

Final Report Writeup
Sam Jenkins

Algorithm

The algorithm that I implemented is called Structural Correspondence Learning, or SCL. More specifically, I implemented an algorithm called SCL-MI, which is an extension of SCL which incorporates mutual information, as described in *Biographies, Bollywood, Boom-boxes and Blenders: Domain Adaptation for Sentiment Classification* [1]. This algorithm is meant to assist in natural language processing tasks where there is little data available for the target task. Consider the example of sentiment analysis for products on Amazon. Suppose we trained a classifier that would take as input the words in a review for a product, and output a label of "positive" or "negative", indicating whether the reviewer had left a positive review or not. Furthermore, suppose that there are many reviews for a certain type of product, but few reviews for a different product. Ideally, we would like to be able to train a classifier on the product with many reviews, and then transfer that knowledge to a classifier that will be able to classify the other product's reviews with good accuracy. Let us call the product with many reviews the source domain, and the product with fewer reviews the target domain. The problem with just training a classifier on the source and using it on the target is that they may have greatly differing vocabularies. For example, a review for a book may have words such as "exciting" or "riveting" correlated with a positive review, while a product such as a kitchen may have words such as "well-designed" or "beautiful" correlated with a positive label. The key idea of the SCL algorithm revolves around the use of what the authors refer to as pivots. Pivots are features that occur in both the source and target domain with a high frequency, and are correlated with each other in unlabeled example. For example, "exciting" in book reviews and "well-designed" in kitchen reviews may appear in large numbers along with the word "good". By this logic, we may decide that when the classifier that was trained on the "books" dataset sees the word "well-designed" in an example from the kitchen dataset, it should treat it as if it were the word "exciting" in a book review. In this case, "good" is the pivot feature, and "exciting" and "well-designed" are domain specific features. Based on this understanding of pivot features, the SCL algorithm is defined as follows [2]:

1. From the unlabeled target and source data, choose m pivots.

2. For each of these m pivots, create a binary classifier $p_m(x)$, which takes in a binary feature vector x representing the words in an example, and outputs a prediction of whether or not pivot m occurs in that example.

3. For each binary classifier p, minimize the weights of the classifier according to

$$w_p = \min_w \sum L\big(w \cdot x,\ p_m(x)\big) + \lambda||w||^2$$

Where λ is a constant of approximately 10^{-4}, x is an unlabeled example from either the target or source domain, and L(w•x, p(x)) is the modified Huber loss function, according to Ando and Zhang [3], which is given by

$$L(p, y) = \begin{cases} \max(0, 1 - py)^2 & if\ py \geq -1 \\ -4py & otherwise \end{cases}$$

where p is the predicted label and y is the true label.

4. Given the weight vectors w_1 through w_m, construct a matrix W in which column m is w_m, the weight vector of the pivot predictor for pivot m. The idea is that each of these columns represent the likelihood of pivot m appearing in an example given feature l: ie if entry m,l in matrix W is positive, then the pivot feature is likely to appear in the example if feature l appears in the example.

5. From this matrix W, it is possible to create m new features, which would be the pivot prediction features. However, this quickly becomes computationally intractable as the number of pivots increases. Instead, a lower dimensional representation of the pivot predictors is computed through a singular value decomposition of W. Given:

$$W = UDV^T ,$$

we let $\theta = U^T$, so that θ becomes the top left singular vectors of W, which are the representations of the pivot predictors in the lower dimensional space.

6. Finally, we train the actual sentiment predictor. For both training and testing examples, when we see a feature vector x, we compute θx, and append that feature vector to the original feature vector x. The expectation is that if the pivots are well selected, then θx will encode the relationship between domain-specific features that are indicative of the overall label.

The main difference between SCL and SCL-MI is the method of pivot selection. In the original SCL algorithm pivots are selected based on frequency of appearance in both domains. However, the original SCL algorithm was developed for use in part of speech tagging between domains, which did not translate well to sentiment classification. SCL-MI, on the other hand, also selects pivots which have a high mutual information to the source domains class label. Therefore, labeled examples from the source domain are also required for SCL-MI to function.

Another difference between SCL and SCL-MI is the concept of misalignment correction. The SCL algorithm can occasionally produce misalignments between the source and target domain in the case that there are features that are predictive in the source domain, but not in the target domain, and vice versa. In this case, the pivot predictor projection that SCL produces may have projections that are discriminative in the source domain, but not in the target domain. For example, a book review may contain widely diverse features that describe the topic of the book, which can range from politics to fiction to history. While this may be an important feature for the book classification, there is no way to adapt that feature to a review for electronics or a kitchen, as there is no corresponding feature. In this case, SCL-MI uses a small amount of labeled target data (~50 examples) in order to preform misalignment correction. Misalignment correction is given as follows:

1. Suppose we have a classifier that has been trained according to the SCL algorithm described above. We can write the objective function of the classifier as

$$\min_{w,v} \sum_{i} L(w'x_i + v'\theta x_i, y_i) + \lambda||w||^2 + \mu||v||^2$$

where y_i is the label of example x_i, w is the weight vector which weighs the original features x_i, and v is the weight vector which weighs the projected features θx_i, and λ and μ are both constants, suggested by the authors to be 10^{-4} and 0, respectively.

2. Given weight vectors w_s and v_s from the trained classifier above, as well as labeled examples from the target domain, x_j, do:

$$\min_{w,v} \sum_{j} L\left(w'x_j + v'\theta x_j, y_j\right) + \lambda||w||^2 + \mu||v - v_s||^2$$

where λ and μ are both now 10^{-1}.

The intuition behind this process is that the weight vector w is too large for a small amount of target data to significantly influence, but the vector v is much smaller and could be influenced by said target data. By applying this misalignment correcting, we can improve the performance of the classifier on the target domain with only a small amount of target labeled data, which could be feasibly provided by a human observer.

$\mathcal{A}$-Distance Metric

Another concept introduced by this paper is that of an empirical method for calculating a metric known as the $\mathcal{A}$-distance [4]. This metric is intended to be a measure of the difference between two different domains in terms of their classification accuracy when adapting from one domain to the other. The mathematical expression for the $\mathcal{A}$-distance is given by:

$$d_{\mathcal{A}}(\mathcal{D}, \mathcal{D}') = 2 \sup_{A \in A} |\Pr[A]_{\mathcal{D}} - \Pr[A]_{\mathcal{D}'}|$$

Where $\mathcal{D}, \mathcal{D}'$ are two probability distributions. This value can be approximated using the SCL algorithm by performing the following:

1. Compute the SCL pivot matrix between 2 domains.

2. Create a new data set consisting of the adapted features, θx, and the label representing the domain that the corresponding example came from. Train a linear classifier on this data.

3. Compute the observed average Huber loss per example. Subtract this value from 1 and multiply it by 100.

When this value is 0, the two domains are indistinguishable to our classifier, and when it is 100, they are completely different. By computing this metric on our different datasets, we can determine whether adaptation is likely to be helpful between these domains or not.

Implementation

Model selection was based on those used in the source paper. A stochastic gradient descent model was used for the pivot predictors, provided by a sklearn framework. A logistic regression classifier was used for the label classifier, provided by the same. Most of the actual work of the algorithm was done in the preprocessing of data, which included data splitting, pivot selection, and data manipulation, which were all implemented by me. The main library used was sklearn which provided model frameworks, data structures, and analysis tools.

Experiments and Analysis

Because the source paper used the amazon review dataset, I decided to use that dataset as a baseline in order to set the hyperparameters of the algorithm before moving on to other datasets. All experiments compare the performance of three different classifiers trained by the algorithm, on both the source and target domain. The first is the baseline classifier, which is trained on the source dataset and tested on the target dataset with no adaptation. The second is the adapted classifier, which is trained on the source dataset adapted with the new pivot features, and tested on the target dataset adapted with the same. The third classifier is the corrected classifier, which is the adapted classifier with the weights adjusted according to the misalignment correction procedure described above. All experiments where performed with 5-fold stratified cross validation, with a training set of 1600 examples and a testing set of 400 examples. Therefore, the results are averaged over the 5 folds. The first experiment, shown in Figure 1, was

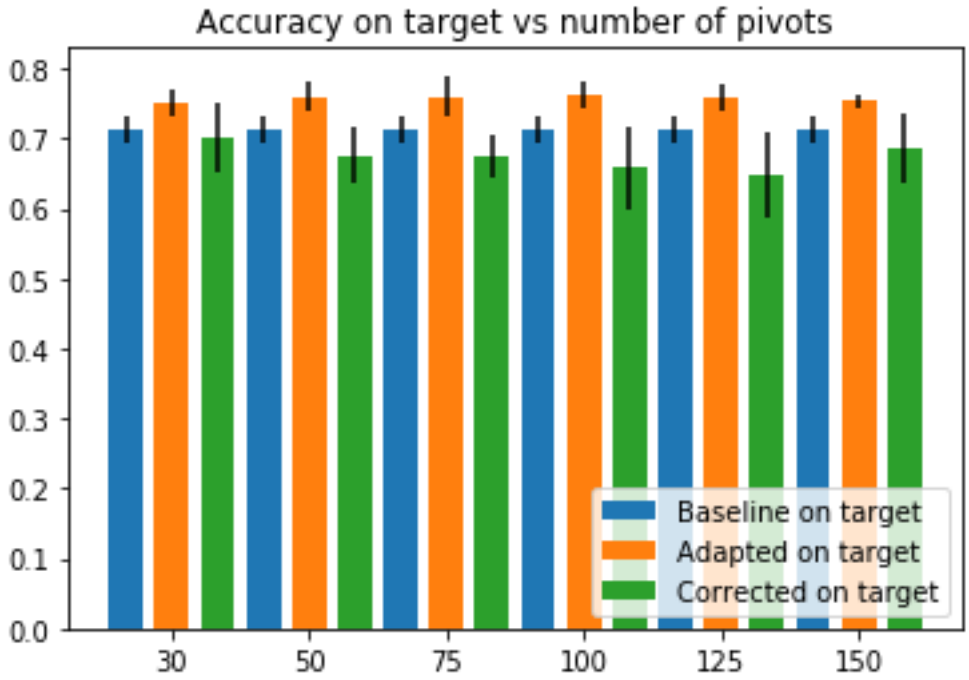


Figure 1

to determine whether the number of pivot features had a significant outcome on the accuracy of the adapted classifier. This experiment was performed Multi-Domain Sentiment dataset, specifically adapting the “dvd” domain to the “kitchen” domain. The dimension of the lower order approximation of the pivot matrix was set to 25 for each run. Additionally, Area under the

Number of Pivots	Baseline	Adapted	Corrected
30	0.773	0.832	0.778
50	**0.773**	**0.841**	**0.765**
75	0.773	0.825	0.745
100	0.773	0.829	0.736
125	0.773	0.827	0.713
150	0.773	0.836	0.759

ROC curve was also collected, which is shown in

Table 1

Table 1. According to these results, there did not appear to be any statistically significant increase in accuracy over the number of pivots, so 50 pivots was chosen as a basis for all further experiments, as it had the highest area under the ROC. The second experiment performed was to determine whether the dimensionality of the pivot matrix representation had a significant effect on the accuracy. The results of the accuracy and area under ROC can be seen in Figure 2 and Table 2, respectively.

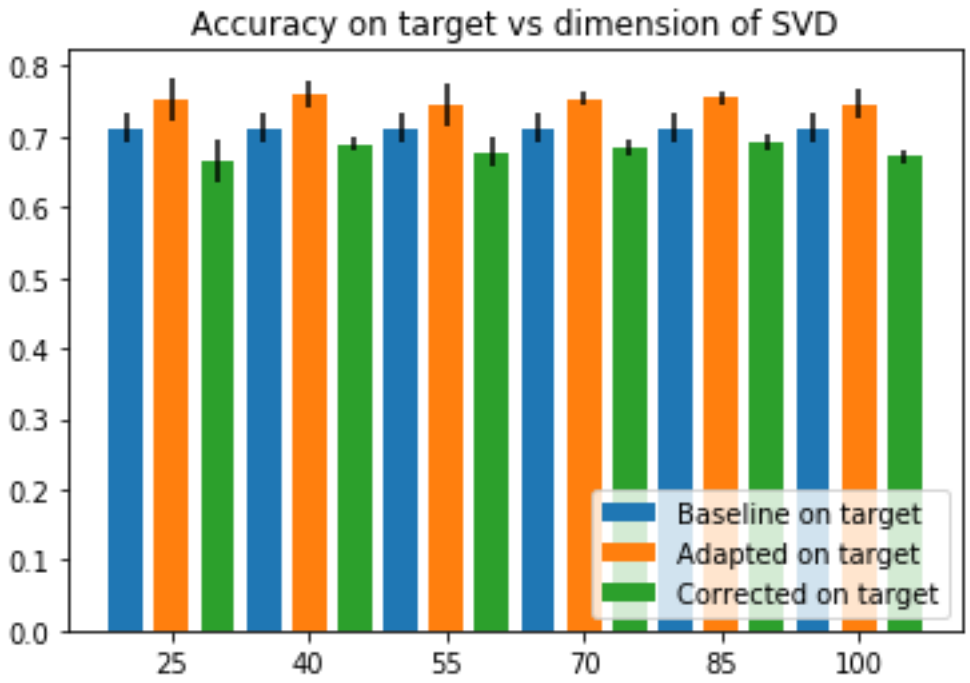


Figure 2

Again, there does not to be any statistically significant increase in the accuracy based on the dimension of the SVD reduction, so a dimension of 40 was chosen for further experiments based

on the fact that it had the highest area under the ROC and that it would be faster to compute than `a more complex SVD.

SVD Dimension	Baseline	Adapted	Corrected
25	0.773	0.825	0.737
40	**0.773**	**0.836**	**0.756**
55	0.773	0.821	0.734
70	0.773	0.825	0.744
85	0.773	0.826	0.754
100	0.773	0.815	0.733

Table 2

Based on these first two experiments, the hyperparameters of number of pivots and dimensionality of the pivot matrix reduction were chosen to be 50 and 40, respectively. Further experiments were then performed on the 20-Newsgroups dataset and the ECML/PKDD task A dataset. A table comparing their accuracies and the accuracies of the "Dvd" and "kitchen" adaptation can be seen in Table 3. Table 4 contains a comparison of the area under their ROCs.

Dataset	Baseline	Adapted	Corrected
Multi-Domain Sentiment	0.685 ± 0.03	0.732 ± 0.03	0.686 ± 0.02
20-Newsgroup	0.801 ± 0.02	0.751 ± 0.01	0.781 ± 0.02
ECML/PKDD	0.956 ± 0.01	0.912 ± 0.02	0.828 ± 0.13

Table 3

Dataset	Baseline	Adapted	Corrected
Multi-Domain Sentiment	0.743	0.815	0.755
20-Newsgroup	0.883	0.837	0.864
ECML/PKDD	0.991	0.975	0.955

Table 4

Performing t-tests between the classifiers on the different datasets allows us to conclude whether or not the changes made by the SCL algorithm offer an improvement over the baseline classifier. Table 5 shows the 95% confidence intervals for these comparisons

Dataset	Baseline vs Adapted	Baseline vs Corrected	Adapted vs Corrected
Multi-Domain Sentiment	-0.09075 to -0.00325	-0.03818 to 0.03618	0.00882 to 0.08318
20-Newsgroup	0.02694 to 0.07306	-0.00917 to 0.04917	-0.05306 to -0.00694
ECML/PKDD	0.02094 to 0.06706	-0.00646 to 0.26246	-0.05164 to 0.21964

Table 5

Conclusions

Based on the results in Table 5, we can conclude whether the SCL algorithm is statistically likely to improve performance over the baseline based on the dataset. On the Multi-Domain Sentiment dataset, because the t-test interval between adapted and baseline is negative, we can conclude that the SCL algorithm has improved performance over the baseline. However, there is no significant difference between the baseline and corrected classifiers, and the corrected classifier performs worse than the adapted classifier. The performance of the adapted classifier agrees with the performance shown in the source paper, but that of the corrected classifier does not. This may be due to an incorrect implementation, but given the explanation in the paper, I am fairly certain that the misalignment correction procedure is working correctly, so I am not sure where other error may have been introduced into the program. However, the base SCL-MI algorithm preformed as expected. On the 20-Newsgroup dataset, however, the opposite is true. The baseline classifier actually performed better than the adapted classifier. The corrected classifier also performed better than the adapted classifier, but I do not believe that any useful results can be drawn from that based on the high chance of errors being present in the misalignment correction procedure. However, the fact that the adapted classifier did not perform well on the 20-Newsgroup is interesting. This may be due to the fact that there is a high degree of similarity between the source and target domains for this dataset. Since the classification task between high level domains such as “computer” and “science” may have significant overlap in the feature space, the addition of the adapted features between the domains may have obfuscated the actually important information. The whole point of the SCL algorithm is that it allows the classifier to correlate words that are not in both the source and domain with words that are in both the source and domain, as well as possessing a high degree of information about the class label. When the vocabulary between the source and target is very similar, this strategy does not appear to work very well. A similar situation was observed on the ECML/PKDD dataset. Again, the baseline classifier performed better than the adapted classifier, and was not significantly different than the corrected classifier. Because there as a high correlation between the features in the test and train datasets that had high information relative to the label, the remaining words that do not occur in both vocabularies would have very little information relative to the feature, so by extracting the pivot features (i. e. features with high mutual information to the label), and adapting the remaining features, it is possible that some information was lost. This suggests to me that the SCL algorithm is best employed when there is a significant difference between the source and target domain.

Further Reading

As the SCL algorithm was developed in 2007, there have been numerous extensions and adaptations of its ideas since then. Once such recent extension is detailed in *Neural Structural Correspondence Learning for Domain Adaptation* [5]. The authors of this paper aimed to marry the more recent use of neural networks, and in particular autoencoders, for domain adaptation, with the SCL algorithm. In the paper, they propose two novel neural network architectures that combine the concepts of autoencoders and SCL: Autoencoder SCL (AE-SCL), and Autoencoder SCL with Similarity Regularization (AE-SCL-SR). In the AE-SCL model, the structure is given as follows:

1. Given an example X, with feature vector x, strip the pivot features out of x to get a new feature vector x^{np}.

2. Encode the new feature vector x^{np} into a lower dimensional representation through the use of a sigmoid activation function.

3. Train a classifier on the lower dimensional representation to predict the occurrence of pivot features, x^{p}, in the original feature vector x.

4. Similar to the SCL algorithm, the original feature vector x is then adapted with the pivot predictor, the result of which is then concatenated with the original x and fed into the sentiment classifier.

This structure aims to take advantage of the fact that autoencoders strip out unessential features when transforming the data into a lower dimension representation, which allows the resultant pivot predictor transform to more accurately reflect the actually important features in the classification problem. AE-SCL-SR is a further improvement on this model, taking advantage of an interesting observation about pivot appearances. There are many pivots that, although they are different words, convey the same information about the label. For example, although "great" and "excellent" are different potential features of an example, they both convey approximately the same level of information about the class label, and therefore could be treated the same by the classifier. Furthermore, for two examples that contain these kinds of similar pivots, even if the pivot is not the same, the set of pivots that *do not* occur in the examples is similar. For instance, if one example contained the word "great" and the other example contained the word "excellent", we would not expect the words "terrible", "useless", or "a waste" to appear in *either* of the examples. AE-SCL-SR uses these observations by pre-training the pivot reconstruction matrix with a word embedding model, which is kept fixed throughout the process. Because the reconstruction matrix is fixed, two examples that have pivots with similar meanings will have similar pivot predictions, even if the actual pivots differ. Furthermore, the prediction of pivots that have conflicting meaning with the pivots that do occur will be much lower. In all other respects, AE-SCL-SR is the exact same as AE-SCL. These improvements allow the AE-SCL-SR model to have as much as a 5% increase in accuracy over an MSDA-DAN model, another autoencoder model that has been shown to have good performance on domain adaptation for sentiment classification, as well as up to an 11% increase over a non-adapted classifier. In this way, the SCL algorithm has been shown to be quite useful even after the focus of domain adaptation shifted to neural networks, and is likely to be a powerful technique in the future even with new developments.

Citations

[1] J. Blitzer, M. Dredze and F. Pereira, "Biographies, Bollywood, Boom-boxes and Blenders: Domain Adaptation for Sentiment Classification", 2007. [Accessed 1 December 2019].

[2] J. Blitzer, R. McDonald and F. Pereira, "Domain Adaptation with Structural Correspondence Learning", 2006. [Accessed 2 December 2019].

[3] R. Ando and T. Zhang, "A Framework for Learning Predictive Structures from Multiple Tasks and Unlabeled Data", *Journal of Machine Learning Research*, vol. 6, 2005. [Accessed 3 December 2019].

[4] S. Ben-David, J. Blitzer, K. Crammer and F. Pereira, "Analysis of Representations for Domain Adaptation", 2007. [Accessed 7 December 2019].

[5] Y. Ziser and R. Reichart, "Neural Structural Correspondence Learning for Domain Adaptation", 2017. [Accessed 7 December 2019].

Locally Weighted Ensembling

Kennan LeJeune

December 7, 2019

1 Introduction

Locally Weighted Ensembling is a transductive parameter knowledge transfer framework [1] intended to improve the learning of a target task τ on a testing domain T, transferring models $M_1, M_2, \ldots, M_k$ trained on labeled domains of interest $D_1, D_2, \ldots, D_k$. For any example x, we weight the model predictions according to their performance in the neighborhood of other examples clustered near x, making an overall prediction with a weighted average of the model outputs at x.

1.1 Example

To best illustrate the effectiveness of such a framework, consider a toy problem with datasets D_1, D_2 and respective models M_1, M_2 as shown by Figure 2. Although each training set has a linear decision boundary, the optimal boundary on the test set follows a v-shape. In this context, we have M_1 which can effectively classify the region R_1, and M_2 which can classify R_2. Rather than merging the test domains, or globally weighting the models, we can produce a weight based on the quality of M_i at any example x, so that the test set can weight M_1 highly to classify R_1, but also weight M_2 more highly to classify R_2 when appropriate [2]. Optimally, we would determine this with Bayesian model averaging to compute the posterior distribution of y as $P(y|x)$ [2]. However, in the case of an unlabeled test domain, we are unable to compute $P(M_i|D)$ in order to properly compute

$$P(y|x) = \sum_{i=1}^{k} P(y|x, D, M_i)P(M_i|D) \tag{1}$$

1.2 Estimating Model Output on an Unlabeled Domain

We can estimate $P(y|M_i, x)$ on the test domain by examining the similarity to M_i on examples in the space around x, and examples which are clustered in the test domain around x [2]. Consider graphs $G_M = (V, E_M)$ and $G_T = (V, E_T)$ where $V = \{x \in T\}$. We construct E_T by clustering the T and adding an edge between all pairs of test examples which are members of the same cluster.

Similarly, E_M is constructed by connecting any two examples $u, v \in T$ if a model M predicts the same class for both u and v. We can construct G_{M_i} for every model M_i of a training domain D_i, and G_T based on the clustered test domain [2].

For any x, we can compute the model weight at x, which is proportional to the similarity of its local structures. If we denote the sets of neighbors of $x \in G_M, G_T$ to be V_M, V_T respectively, then the weight calculation [2] is

$$w_{M,x} \propto s(G_M, G_T; x) = \frac{\sum_{v_1 \in V_M} \sum_{v_2 \in V_T} 1\{v_1 = v_2\}}{|V_M| + |V_T|} \tag{2}$$

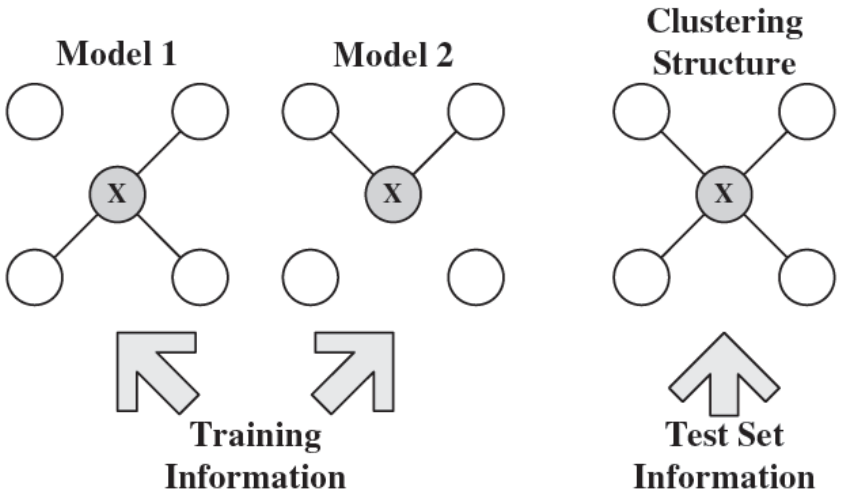


Figure 1: Example Similarity Graph Construction

Figure 1 depicts an example of three such graphs, where $w_{M_1,x} \propto \frac{3}{4}$, and $w_{M_2,x} \propto \frac{1}{2}$ since models 1 and 2 have 3 of 4 and 2 of 4 neighbors in common with x on the test set respectively. With this, we can effectively weight our models to maximize their utility on a per-example basis, providing insight into the power of this framework.

It is crucial to note that this estimation is based upon a clustering assumption [2], asserting that an average clustered train set must be mostly representative of the actual distribution of class labels for examples in the set. If this assumption is not fulfilled, then we simply take an evenly weighted average of the model predictions for all $x \in T$ such that

$$P(y|x) = \sum_{i=1}^{k} w_{M_i,x} P(y|x, M_i) \tag{3}$$

where $P(y|x, M_i)$ denotes the output prediction of M_i for example x, and use $w_{M_i,x} = 1/k$ for all models.

Figure 2: Toy Problem

2 Algorithm Implementation

Figure 3 outlines the general structure which was followed for the algorithm implementation.

Algorithm: Locally Weighted Ensemble (LWE)
Input: (1)A training set D or k training sets $D_1, \ldots, D_k$
(2)k classification models $M_1, \ldots, M_k$ $(k > 1)$
(3)A test set T which comes from a different domain but the classification task is the same.
(4)A threshold δ and cluster number c'.
Output: The set of predicted labels Y for examples in T.
Algorithm:

1. Perform clustering on the training set(s), **IF** the average purity of clustering is less than 0.5, set $w_{M_i,\mathbf{x}} = \frac{1}{k}$ for all M_i and $\mathbf{x}$, and compute the posterior using Eq.(7) for each $\mathbf{x} \in T$. **RETURN**.
2. Group test examples into c' clusters and construct neighborhood graphs based on the clustering results and all the k models. Set $T' = \Phi$.
3. **FOR** each $\mathbf{x} \in T$,
   - **FOR** each model M_i, compute the model weight $w_{M_i,\mathbf{x}}$ according to Eq.(5).
   - **IF** $s_{\text{avg}}(\mathbf{x}) \geq \delta$, decide $\mathbf{x}$'s label based on the weighted ensemble's output $P(y|E,\mathbf{x})$ obtained using Eq.(7), **ELSE** put $\mathbf{x}$ into T'.
4. **FOR** each $\mathbf{x} \in T'$, predict $\mathbf{x}$'s label from the "unsupervised" classifier U, i.e., estimate $P(y|U,\mathbf{x})$ using Eq.(11) or Eq.(12). **RETURN**.

Figure 3: Locally Weighted Ensembling Framework

We begin by training models M_i corresponding to each training set D_i with Logistic Regression, and pass each trained model to the function. We perform clustering on the each training set D_i using `AgglomerativeClustering` from scikit-learn, and compute the confusion matrix according to the class labels for each $x \in D_i$. If the clustering assumption fails, then we cannot estimate the structure of the test domain using the structure of the training domains, so we return the average model output as demonstrated in Equation (3). Otherwise, we proceed with the LWE algorithm. We cluster the testing domain and then construct neighborhood graphs as proposed in Section 1.2.

Using these neighborhoods, we iterate over the examples in the test domain and compute a

normalized model weight [2],

$$w_{M_i,x} = \frac{s(G_{M_i}, G_T; x)}{\sum_{i=1}^{k} s(G_{M_i}, G_T; x)} \tag{4}$$

and if the the average $s_{avg}(x) = \frac{1}{k}\sum_{i=1}^{k} s(G_{M_i,G_T;x})$ satisfies a threshold value δ, compute the output for x using Equation (3). Otherwise, estimate the output of x by taking the average output of neighbors of x which are members of the same cluster, and also have an average structural similarity $s_{avg}(x) \geq \delta$.

This step allows us to approximate any example with reasonable accuracy regardless of a lack of confident model output, and is a significant portion of the manner in which the framework distinguishes itself from other ensembling methods, allowing us to leverage both structural similarity and model confidence for local regions of the test space.

We have additionally implemented a baseline classifier (SGA) [2] for comparison which simply uses the weighted output from Equation (3) with $w_{M_i} = 1/k$ without the clustering assumption, and a variant pLWE, which uses Equation (3) with the local model weight, regardless of structural similarity at x. We compare these approaches in Section 3.

3 Results and Analysis

We first run the algorithm on the Multi-Domain Sentiment Dataset, consisting of Amazon Reviews. We apply training on DVD reviews to produce sentiment classification for reviews of Kitchen products, training on 1000 reviews from a single dataset and test on a set of 300 reviews, allowing for a 5000 dimensional feature space.

DVD to Kitchen	**SGA**	**LWE**	**pLWE**
Avg. Accuracy	0.7506666667	0.774	0.7673333333
St. Dev	0.01404753834	0.01311487705	0.01535195

Figure 4: Amazon Review Sentiment Classification

While the classifier's accuracy is not particularly stellar, LWE manages to significantly outperform the baseline classifier, SGA, with the true mean of the difference between LWE and SGA's accuracy $\mu \in [0.0033, 0.0427]$ with 95% confidence. However, pLWE fails to be statistically significant, with $\mu \in [-0.00447, 0.03647]$ with 95% confidence, despite overall poor clustering quality on both the training and testing domains.

Next, we run the algorithm on ECMLL/PKDD Task A and Task B, spam datasets, as described in the group paper, with results shown in Figure 5.

Task A has 4000 source domain examples, resulting in a high initial baseline, which is only slightly improved upon by LWE and pLWE, although the differences are weakly statistically significant.

ECML/PKDD Task A	SGA	LWE	pLWE
Avg. Accuracy	0.9476666667	0.9553333333	0.9513333333
St. Dev	0.002516611478	0.001527525232	0.003785938897

Figure 5: Spam: ECMLL/PKDD Task A Results

ECML/PKDD Task B	SGA	LWE	pLWE
Avg. Accuracy	0.8733333333	0.9066666667	0.8966666667
St. Dev	0.01154700538	0.0132231652	0.0144

Figure 6: Spam: ECMLL/PKDD Task B Results

Task B has 15 sets containing 100 examples each, yielding a lower base accuracy, since not all models are equally effective when training across multiple domains, yielding an inferior approach when predicting with the averaged output in Equation (3). The results of LWE and pLWE are both statistically significant when compared against the base classifier as demonstrated in the group writeup.

Finally, we run on the 20 Newsgroups dataset [2], yielding the best overall performance of this algorithm as demonstrated by results in Figure 7.

20 Newsgroups	SGA	LWE	pLWE
Avg. Accuracy	0.8186666667	0.975	0.9276666667
St. Dev	0.003511884584	0.01311487705	0.03453018004

Figure 7: 20 Newsgroups Results

We find this dataset to be optimal for this approach since it is extremely purely clustered, but also has substantial depth to produce highly confident classifiers for nearly any local $x \in T$, although weaker points still exist near decision boundaries, as indicated by differing accuracy between results on LWE and pLWE. The accuracy of LWE outpaces the baseline by nearly 16%, and demonstrates one of the most effective applications of this framework in this problem space.

4 Comparison to Other Relevant Literature

We have explored related work detailing with an alternative approach to transform the model of training examples to produce a Bayesian prior which can be applied to the test domain [3]. Other approaches alternatively attempt to reduce covariate shift by reweighting training examples according to their ratio of test frequency to train frequency to minimize the reweighted probability [4]. In each case, the methods fail to consider multiple source domains, and lose out on the benefit of model localization on a test domain.

5 Conclusions

We have implemented the Locally Weighted Ensembling framework, in addition to a Partially Locally Weighted variant, and a baseline classifier on which to compare performance across three datasets. In each case, the LWE framework proved to provide statistically significant performance gains, approaching 16% accuracy improvement on some datasets.

With this framework, we effectively outperform a standard classifier in cases where source and target domains differ, or match in cases where we deal with a standard machine learining problem. Finally, we have demonstrated the effectiveness of this approach to problems where target domain data is limited, but plentiful in other related areas, and demonstrated its utility for future use.

References

[1] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, 1345–1359, 2010. DOI: 10.1109/tkde.2009.191.

[2] J. Gao, W. Fan, J. Jiang, and J. Han, "Knowledge transfer via multiple model local structure mapping," *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 08*, 2008. DOI: 10.1145/1401890.1401928.

[3] X. Li and J. Bilmes, *A bayesian divergence prior for classifier adaptation*, 2007.

[4] H. Shimodaira, "Improving predictive inference under covariate shift by weighting the log-likelihood function," *Journal of Statistical Planning and Inference*, vol. 90, no. 2, 227–244, 2000. DOI: 10.1016/s0378-3758(00)00115-4.

Self-Taught Clustering

Chris Toomey

Papers Read

For the domain of transfer learning, I implemented a self-taught clustering, based on the paper "Self-taught Clustering".[1] In order to have more background in the area, I first read the paper "Self-taught Learning: Transfer Learning from Unlabeled Data".[2] The paper essentially attempts to find a higher level feature representation on a set of auxiliary data that they then apply to the learning task, so the classifier does not need to learn the feature representation itself. They make the distinction between self-taught learning and transfer learning by specifying that transfer learning is typically used for similar domains (e.g. classifying unlabeled images of elephants and rhinos by first classifying labeled images of horses and goats), whereas self-taught learning does not need to be a similar domain, only the same preliminary data representation (e.g. images).

Self-taught Learning

For self-taught learning specifically, the most important aspect is being able to learn how to represent the data. In order to do so, the authors modify an optimization problem proposed earlier:

$$minimize_{b,a} \sum_i ||x_u^{(i)} - \sum_j a_j^{(i)} b_j||_2^2 + \beta||a^{(i)}||_i s.t. ||b_j||_2 \leq 1, \forall j \in 1, 2, ..., s \quad (1)$$

where b is set of basis vectors $\{b_1, b_2, ..., b_s\}$ where $b_j \in \setminus^n$, a is the set of activations $\{a^{(1)}, a^{(2)}, ..., a^{(k)}\}$ where $a^{(i)} \in \setminus^s$, β is a trade-off value, and $a_j^{(i)}$ is the activation of basis b_j for the input $x_u^{(i)}$. Essentially, the objective function uses the two terms in the optimization problem above by: using the first term to be reconstructed as a linear combination of the basis b_j and using the second term to make sure the activations are as sparse as possible. This sparsity is important because it will allow for more generalization when applied to the target domain. Solving this for the unlabeled training data will give the basis vectors. The authors then propose applying a similar optimization problem to a set of labeled training data, with the intention this time to learn the activations for the basis vectors:

$$\hat{a}(x_l^{(i)}) = argmin_{a^{(i)}} ||x_l^{(i)} - \sum_j a_j^{(i)} b_j||_2^2 + \beta||a^{(i)}||_1 \quad (2)$$

This ensures the data is represented as a sparse linear combination of the input $x_l^{(i)}$, and a classifier can be trained on this data to find the activation set $\hat{a}$. The classifier is then returned and can be applied to a different domain with unlabeled examples.

This area is relatively novel in the context of transfer learning. Unsupervised transfer learning is not as studied as inductive and transductive. The difficulty in this area is finding a useful representation for the data without using labels. Self-taught learning can be extended in the area of clustering, known as self-taught clustering.

Self-taught Clustering

Self-taught clustering is an instance of unsupervised learning. It uses a target data set X, an auxiliary data set Y, and a common feature space between X and Y, Z. The output is the optimal clustering of the target data set. The sets $\hat{X}, \hat{Y}, \hat{Z}$ are used to describe the sets of clusters for each mapping (for example, $\hat{x}_i$ is the i$^{\text{th}}$ cluster of data points belonging to X). We need a clustering function for each variable, C_X, C_Y, and C_Z to map data to a specific cluster (e.g. $C_X : X \rightarrow \hat{X}$. In order to do this, it uses the auxiliary data set and common feature space in order to improve its clustering algorithm at each iteration. The authors use an extension of information theoretic co-clustering to aid them in their self-taught clustering algorithm. The objective function attempts to minimize the loss in mutual information between the examples and features:

$$I(X, Z) - I(\hat{X}, \hat{Z}) \tag{3}$$

where I denotes the mutual information between the instances and features. Both these can be simplified further:

$$I(X, Z) = \sum_{x \in X} \sum_{z \in Z} p(x, z) log(\frac{p(x, z)}{p(x)p(z)}) I(\hat{X}, \hat{Z}) = p(\hat{x}, \hat{z}) = \sum_{x \in \hat{x}} \sum_{z \in \hat{z}} p(x, z) \tag{4}$$

The self-taught clustering algorithm uses co-clustering to improve the target clusters, meaning the auxiliary data Y is being used to help improve the clustering of X. The objective function is therefore defined as

$$\tau = I(X, Z) - I(\hat{X}, \hat{Z}) + \lambda \left[I(Y, Z) - I(\hat{Y}, \hat{Z}) \right] \tag{5}$$

The mutual information involving Y and $\hat{Y}$ is multiplied by a trade-off parameter, λ, in order to make sure the auxiliary data does not have too much influence over the target data. Minimizing the objective function is non-trivial, since it is non-convex. The authors therefore used the Kullback-Leibler divergence (KL divergence) in order to re-write the equation and try to minimized it.

In order to minimize this function, start by creating the joint probability distribution of a variable and the feature space with respect to their co-clusters:

$$\hat{p}(x, z) = p(\hat{x}, \hat{z}) p\left(\frac{x}{\hat{x}}\right) p\left(\frac{z}{\hat{z}}\right) \tag{6}$$

where $p(\hat{x}, \hat{z})$ is the joint probability distribution of clusters $\hat{x}, \hat{z}$, $p\left(\frac{x}{\hat{x}}\right)$ is the total number of features in x divided by the total number of features in the cluster $\hat{x}$, and $p\left(\frac{z}{\hat{z}}\right)$ is the total number of times the features z appears divided by the total number of times each feature in $\hat{z}$ appears. The same equation can be written in terms of Y to define the co-cluster joint probability distribution between Y and Z:

$$\hat{q}(y, z) = q(\hat{y}, \hat{z}) q\left(\frac{y}{\hat{y}}\right) q\left(\frac{z}{\hat{z}}\right) \tag{7}$$

The objective function can be re-written in the form of KL divergence:

$$\tau = I(X, Z) - I(\hat{X}, \hat{Z}) + \lambda \left[I(Y, Z) - I(\hat{Y}, \hat{Z}) \right] \tag{8}$$

$$= D(p(X, Z) || \hat{p}(X, Z)) + \lambda D(q(Y, Z) || \hat{q}(Y, Z)) \tag{9}$$

where $D(\ ||\ )$ denotes the KL divergence between the probability distributions,

$$D(p(x), q(x)) = \sum_{x} p(x) log\left(\frac{p(x)}{q(x)}\right) \tag{10}$$

The KL divergence can be simplified for our objective function:

$$D(p(X, Z) || \hat{p}(X, Z)) = \sum_{\hat{x} \in \hat{X}} \sum_{x \in \hat{x}} p(x) D(p(Z|x) || \hat{p}(Z|\hat{x})) \tag{11}$$

$$= \sum_{\hat{z} \in \hat{Z}} \sum_{z \in \hat{z}} p(z) D(p(X|z) || \hat{p}(X|\hat{z})) \tag{12}$$

which can be applied similarly for Y:

$$D(p(Y, Z) || \hat{p}(Y, Z)) = \sum_{\hat{y} \in \hat{Y}} \sum_{y \in \hat{y}} p(y) D(p(Z|y) || \hat{p}(Z|\hat{y})) \tag{13}$$

$$= \sum_{\hat{z} \in \hat{Z}} \sum_{z \in \hat{z}} p(z) D(p(Y|z) || \hat{p}(Y|\hat{z})) \tag{14}$$

So, by minimizing $D(p(Z|x) || \hat{p}(Z|\hat{x}))$ for a single x, the optimization function will find a global minimum and the values of x will be put into their respective clusters. So, for each data point x, the clustering function C_X is defined as:

$$C_X(x) = argmin_{\hat{x} \in \hat{X}} D(p(Z|x) || \hat{p}(Z|\hat{x})) \tag{15}$$

and for each data point y:

$$C_Y(y) = argmin_{\hat{y} \in \hat{Y}} D(p(Z|y)||\hat{p}(Z|\hat{y})) \tag{16}$$

and finally for a feature z:

$$C_Z(z) = argmin_{\hat{z} \in \hat{Z}} p(z) D(p(X|z)||\hat{p}(X|\hat{z})) + \lambda q(z) D(p(Y|z)||\hat{p}(Y|\hat{z})) \tag{17}$$

By finding the $\hat{x}$, $\hat{y}$, and $\hat{z}$ that are best for each data point x, y, and z respectively, the objective function will be minimized for each iteration. Formally, the algorithm implemented can be drawn up as such:

Algorithm Implemented

Algorithm 1: Self Taught Clustering Algorithm

Input: A target unlabeled data set X, an auxiliary unlabeled data set Y, the feature space shared by both X and Y, the initial clustering functions $C_X^{(0)}$, $C_Y^{(0)}$, and $C_Z^{(0)}$, and the number of iterations T

Output: The clustering function $C_X^{(T)}$

Procedure:

1: Initialize the joint probability distributions $p(X, Z)$ and $q(Y, Z)$ based on the data

2: Initialize $\hat{p}^{(0)}(X, Z)$ and $\hat{q}^{(0)}(Y, Z)$

3: for $i = 1...T$ iterations:

4: Update $C_X^{(i)}(X)$, $C_Y^{(i)}(Y)$, and $C_Z^{(i)}(Z)$

5: Update $\hat{p}^{(i)}(X, Z)$ and $\hat{q}^{(i)}(Y, Z)$

6: end for

7: Return $C_X^{(T)}$

I implemented the above algorithm for my final project. I did so using Python, with the following libraries: the log function from math, a few mathematical functions from numpy, shuffle from random, several functions from os in order to extract the data, a few data types from typing, again to help extract the data, copy in order to deepcopy the contents of a list to another, the standard deviation function from statistics, and sys in order to read the arguments.

Experiments

For each data set, I used 25 examples for the target domain, with 50 total features, 10 iterations, and a trade-off of 0.05. I adjusted the examples from the source domain slightly as this affected the entropy and accuracy of the data. I used four metrics to measure how well the algorithm performed: entropy (which is what the authors from the paper used), accuracy, precision, and recall.

Multi-domain sentiment dataset

The first data set I ran on was the multi-domain sentiment dataset, where the "dvd" data set was the source domain and the "kitchen" data set was the target domain. I used 75 examples in the source domain. The starting entropy for a random assignment was 0.352, whereas the start accuracy was 0.48, start precision was 0.5, and recall was 0.462. After one iteration, the entropy dropped to 0.177, and the accuracy was 0.48. The precision was 1.0, whereas the recall was 0.48. Unfortunately, these metrics do not improve with more iterations, so I was unable to get the algorithm fully working. Because these did not change, the variance barely changed for each of the metrics. The accuracy standard deviation was 0.0, the precision standard deviation was 0.158, and the recall standard deviation was 0.006.

Newsgroup

Next, I ran my algorithm on the newsgroups dataset. The results were very similar. I increased the examples from the source domain slightly, to 125. The entropy was 0.325 to start, while the accuracy was 0.52, the precision was 0.533, and the recall was 0.615. After the first iteration, the entropy fell to 0.147, the accuracy became 0.6, the precision rose to 1.0, and the recall was 0.6. These numbers stayed consistent throughout each iteration. The end standard deviations for accuracy, precision, and recall respectively were 0.025, 0.148, and 0.005.

ECML/PKDD

Finally, I ran the algorithm on the ECML/PKDD dataset. I used 100 examples for the source domain. The starting entropy of the clusters was 0.293. The accuracy was 0.6, and the precision and recall were both 0.615. After one iteration, the entropy drops to 0.169, while the accuracy and precision both drop as well, to 0.56 and 0.15 respectively, while the recall increases to 1.0. After the second itertation, however, the data stays the same, with an entropy slightly increasing to 0.177, an accuracy of 0.48, and precision and recall of 0. The end standard deviations for accuracy, precision, and recall respectively were 0.043, 0.195, and 0.352 respectively.

Data set	Entropy	Accuracy	Precision	Recall
Multi-domain sentiment	0.177	0.48	1.0	0.48
Newsgroups	0.147	0.6	1.0	0.6
ECML/PKDD	0.177	0.48	0.0	0.0

The above table illustrates the metric for each dataset at the very last iteration. In general, the algorithm does not perform better than randomly guessing according to the metrics used. The self-taught clustering algorithm implemented in the paper performed significantly better than mine, as I could not quite get mine to work. Though they did not use accuracy, precision, or recall, they used it to cluster images from different domains, and on average, their entropy was 0.610. The entropy the paper had was lower on average than all other evaluation method they tested against, especially when only using one target and source domain. The value of the trade-off parameter in my implementation did not seem to matter, whereas in the paper the entropy decreased as the value increased. Similarly, an increase in the number of feature clusters did not seem to affect the entropy of my implementation, but in the paper, the more feature clusters, the worse the entropy.

Conclusion

My implementation of self-taught clustering did not work very well. Though it did cluster to some extent, the clustering classification did not perform well, as evident by my metrics. Unsupervised transfer learning proved to be more difficult than I initially realized. The authors are able to conclude that self-taught clustering represents data better than other clustering methods. My analysis was unable to conclude the same thing. Much of the difficulty occurred in understanding the semantics of some areas in the paper. For example, not understanding where they derived $\frac{p(z)}{p(\hat{z})}$ (which I changed to $p(\frac{z}{\hat{z}})$) pushed back my implementation until I was able to understand it, as they treated the probability as trivial to understand and did not explain the derivation. KL divergence also proved to be more difficult to utilize than I thought. Though the findings of my paper do not necessarily support it, self-taught clustering is an interesting implementation of unsupervised transfer learning.

Citations

[1] W. Dai, Q. Yang, G. Xue, and Y. Yu, "Self-taught clustering," in Proceedings of the 25th International Conference of Machine Learning. ACM, July 2008, pp. 200–207

[2] Raina, R., Battle, A., Lee, H., Packer, B., & Ng, A. Y.(2007). Self-taught learning: transfer learning from unlabeled data. Proceedings of the Twenty-fourth International Conference on Machine Learning (pp.759–766).

Dimensionality Reduction in Transfer Learning

Arthur Xin, sxx132

1 Introduction

Transfer Learning addresses the problem of needing to use existing labelled data to apply to a new group of related data that has related but different characteristics. The usage of dimensionality reduction allows us to learn a low-dimensional latent feature space where the distributions of the source and target data are similar. This latent space provides us a gateway to connect the information between the source data and target data and facilitate knowledge transfer. This way we can then apply traditional learning methods to train models in our latent space that will predict labels in the target space.

2 Background

2.1 Dimensionality Reduction

Dimensionality is a commonly used concept in the data science community. Although considered generally an already thoroughly researched topic, we wanted to discuss differences in different dimensionality reduction methods that will impact our algorithm differently. The first and most widely known technique is principle component analysis (PCA). This technique alone is a powerful tool that projects a dataset into a lower dimensionality in the directions of the greatest variance of while preserving some features of the original dataset. However, this alone cannot guarantee that the distributions of the original dataset will be similar in the reduced latent space. As a result, we cannot use PCA alone to solve a transfer learning problem.

A more recent dimensionality reduction method was developed called maximum variance unfolding (MVU) (Weinberger, Sha, and Saul 2004). It preserves the local distances between neighboring datapoints while having the advantage of PCA in that it preserves the maximum variance for a low-dimensional representation of the data. The MVU applies PCA to an estimated kernel matrix K and selects the first few singular values as the bases and projects the original data in the reduced kernel bases. This way, we can still preserve critical features about the original data and use them for transferring information from the source dataset into our target dataset. In order to solve our problem of estimating the distance between different distributions of the source and domain

2.2 Maximum Mean Discrepancy

In order to solve our problem of estimating the distance between different distributions of the source and target domains, we need a non-parametric estimate criterion. The Maximum Mean Discrepancy (MMD) solves this issue and is defined by

$$Dist(\mathrm{X},\mathrm{Y}) = \sup_{\|f\|_{\mathcal{H}} \leq 1} \left(\frac{1}{n_1}\sum_{i=1}^{n_1} f(x_i) - \frac{1}{n_2}\sum_{i=1}^{n_2} f(y_i)\right)$$

Where *H* is a universal Reproducing Kernel Hilbert Space. Moreover, the Dist(X,Y) is nonnegative and vanishes as n_1 and n_2 approach infinity.

2.3 Gaussian Process

Another potential usage of knowledge transfer is Adaptive Transfer learning using Gaussian Process (Cao et al 2008). Advantages of this algorithm would be that it also includes the priors and hyperparameters of the trained models. Whereas the MVU would be a point estimator of the transfer kernel from the source domain to the target domain, the Gaussian Process would instead be a Bayesian estimation of the previously learned tasks in the kernel as opposed to developing new ones. As a result, the target of the algorithm would be to optimize the conditional distribution:

$$p(\mathbf{y}^{(\mathcal{T})}|\mathbf{y}^{(\mathcal{S})}, \mathbf{X}^{(\mathcal{T})}, \mathbf{X}^{(\mathcal{S})})$$

Where y is the label and X is the dataset, s is source and t is target domains

3 Methodology

3.1 Notation

Term	Description
$x_{src_i} \in \mathbb{R}^m$	The datapoints from the source dataset
y_{src_i}	Labels of the source dataset
$\mathcal{D}_{src} = \{(x_{src_1}, y_{src_1}), \ldots, (x_{src_{n_1}}, y_{src_{n_1}})\}$	The domain for our source dataset, which is a combination of our data along with its labels, of size n_1
$x_{tar_i} \in \mathbb{R}^m$	Datapoints in our target dataset
$\mathcal{D}_{tar} = \{x_{tar_1}, \ldots, x_{tar_{n_2}}\}$	The overall domain for our target dataset, of size n_2
$\psi(X) = X'$	Projection map of X onto a common latent space common to both source and target domains

3.2 Our Algorithm

Input: we have our labeled source data $\mathcal{D}_{src} = \{(x_{src_1}, y_{src_1}), \ldots, (x_{src_{n_1}}, y_{src_{n_1}})\}$ and unlabeled target data, and a positive λ.

Output: $\mathcal{D}_{tar} = \{x_{tar_i}\}$

1. We solve the following semidefinite program (SDP) to obtain a kernel matrix K

$$\begin{aligned} \min_{\widetilde{K}\succeq 0} \quad & \text{trace}(\widetilde{K}L) - \lambda \text{trace}(\widetilde{K}) \\ \text{s.t.} \quad & \widetilde{K}_{ii} + \widetilde{K}_{jj} - 2\widetilde{K}_{ij} + 2\varepsilon = d_{ij}^2, \ \forall (i,j) \in \mathcal{N}, \\ & \widetilde{K}\mathbf{1} = -\varepsilon \mathbf{1}. \end{aligned}$$

2. We apply PCA to the previously determined matrix K to get new representations of our data $\{x'_{src_i}\}$ and $\{x'_{tar_i}\}$ for our original data x_{src} and x_{tar} respectively.
3. We then learn a standard classifier or regressor to predict our labels $f : x'_{src_i} \rightarrow y_{src_i}$
4. We use our previously learned model to predict our labels for the target set D_{tar}. Where $\hat{y}_{tar_i} = f(x'_{tar_i})$
5. Compare our predicted $\hat{y}_{tar_i}$ to the actual labels of y_{tar_i}

3.3 Step 1: Solving the Semi-Definite Programming Problem

In order to have a universal kernel matrix K, we can write it as

$$K = \widetilde{K} + \epsilon I$$

Where $\varepsilon > 0, \widetilde{K} \succeq 0$ is the result of our kernel calculation and I is the identity matrix. This way our kernel will be strictly positive definite so that we can run our calculations without a potential of divide by zero error.

3.4 Step 2: Maximum Mean Discrepancy Embedding

We will now look at how to construct a low-dimension latent space between the source data and the target data. In order to do so, we must minimize the following equation:

$$\text{dist}(X'_{src}, X'_{tar}) = \left\| \frac{1}{n_1} \sum_{i=1}^{n_1} \phi(x'_{src_i}) - \frac{1}{n_2} \sum_{i=1}^{n_2} \phi(x'_{tar_i}) \right\|_{\mathcal{H}}$$

for some $\phi \in \mathcal{H}$. Thus,

$$\begin{aligned} & \text{dist}(X'_{src}, X'_{tar}) \\ & = \text{dist}(\psi(X_{src}), \psi(X_{tar})) \\ & = \left\| \frac{1}{n_1} \sum_{i=1}^{n_1} \phi \circ \psi(x_{src_i}) - \frac{1}{n_2} \sum_{i=1}^{n_2} \phi \circ \psi(x_{tar_i}) \right\|_{\mathcal{H}} \end{aligned} \quad (3)$$

Which can simply be reduced to

$$\text{dist}(X'_{src}, X'_{tar}) = \text{trace}(KL)$$

Where

$$\begin{bmatrix} K_{src,src} & K_{src,tar} \\ K_{tar,src}^T & K_{tar,tar} \end{bmatrix} \in \mathbb{R}^{(n_1+n_2)\times(n_1+n_2)}$$

$$L_{ij} = \begin{cases} \frac{1}{n_1^2} & x_i, x_j \in X_{src}, \\ \frac{1}{n_2^2} & x_i, x_j \in X_{tar}, \\ -\frac{1}{n_1 n_2} & \text{otherwise.} \end{cases}$$

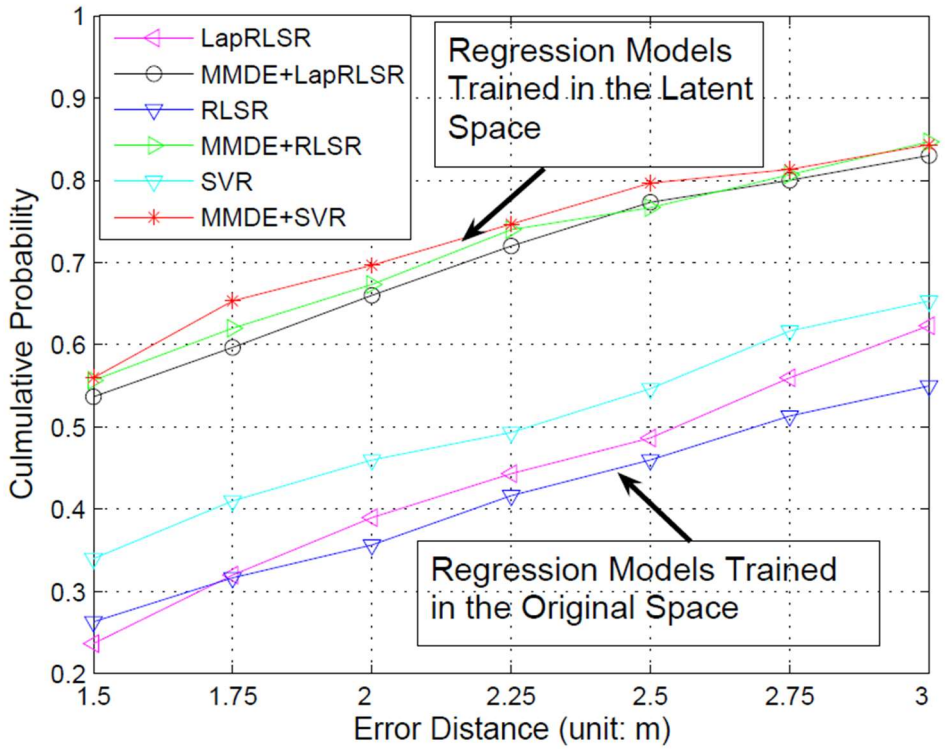
Thus we have essentially reduced our maximum mean discrepancy embedding problem into a kernel problem in which we can solve for any ψ in our projection map.

3.5 Step 3: Apply PCA and an Existing Classifier or Regressor

At this step, we reduce the dimensionality of our kernel using PCA and select the top few eigenvalues that will represent our base for projection. At this point, we can learn any standard classifier. The regression models used in the paper (Pan et al, 2008) includes the Regularized Least Square Regression (RLSR), Support Vector Regression (SVR), Laplacian Regularized Least Square Regressor (LapRLSR).

4 Experimentation

The figure below shows the results produced in their experiments on a WiFi signal strength data.



While I was not able to personally implement this algorithm in our datasets, we see that in the experiment by the authors of this paper (Pan et al 2008) have found that the predictions trained in the latent space had a significantly higher cumulative probability than those trained in the regression models in the original space.

5 Sources

Cao, B.; Pan, S.; Zhang, Y.; Yeung, D.; Yang, Q. 2010. Adaptive Transfer Learning. *In Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence.*

Pan, S.; Kwok, J.; Yang, Q.; 2008. Transfer learning via dimensionality reduction. *In Proceedings of the 23rd national conference on Artificial intelligence*

Weinberger, K. Q.; Sha, F.; and Saul, L. K. 2004. Learning a kernel matrix for nonlinear dimensionality reduction. *In Proceedings of the 21st International Conference on Machine Learning.*

Domain Adaptation Methods for Sentiment Classification with Transfer Learning

David Blincoe, Brendan Dowling, Sam Jenkins,
Kennan LeJeune, Chris Toomey, Arthur Xin
December 7, 2019

Introduction

Traditionally, Machine Learning problems rely on the assumption that the training data and future data domains are in the same feature space and have the same distribution (Pan, Yang). However, a large number of real world applications perform poorly when these assumptions are not met. A common instance of is a classification task on a restricted domain of interest (e.g. classifying malignant brain tumors) with minimal or unlabeled training data, but we have sufficient training data in another domain of interest (e.g. classifying malignant lung carcinomas).

Transfer Learning aims to solve this problem by adapting the knowledge acquired from a training task or training domain for use in a related domain or task. Intuitively, we can apply a solution to a problem to a different but related problem in a humanlike matter.

Definition. *(Transfer Learning)* (Pan, Yang)
Given source and target domains D_S, D_T and learning tasks T_S on the source domain and T_T on the target domain, where we aim to improve the learning of a target predictive function $f_T(\cdot) \in D_T$ using the knowledge in D_S or T_S, where $D_S \neq D_T$ or $T_S \neq T_T$. Note that in the case where both domains and tasks are equivalent, this is analogous to a standard machine learning problem. We can characterize the nature of nearly all Transfer Learning problems by considering three primary cases:

1. $D_S = D_T, T_S \neq T_T$ (Inductive Transfer)
2. $D_S \neq D_T, T_S = T_T$ (Transductive Transfer)
3. $T_S \neq T_T$ and D_S, D_T are unlabeled (Unsupervised Transfer)

In **Inductive Transfer Learning**, the domain of the source and target tasks is the same. In this case, labeled data from the target task must be available in order to *induce* the predictive function that we want the classifier to learn. There are two situations in inductive transfer learning: one where labeled source data is available, and one where it is not. The first scenario is similar to something known as *self-taught learning,* in which the classifier hopes to learn basic patterns from random unlabeled data. The second scenario is similar to multi task learning, where the classifier attempts to learn several classification tasks at the same time, except in this

case we only care about the performance on a single target task, and are hoping to use knowledge learned from the other tasks in order to improve performance on said target task.

Transductive Transfer Learning focuses on an area in which the source and target tasks are the same, but the domains differ. The data used in the target domain is unlabeled, whereas there is an abundance of labeled data in the source domain. Transductive transfer learning can be broken down even further, into two specific scenarios: the first, in which the target and source domains have a different feature space; and the second, where the feature space is the same, but the input data have different marginal probability distributions.

Unsupervised Transfer Learning focuses on a setting where there is no labeled data for either the source and target domains, and the target task is different from the source task. This situation is common in areas such as clustering and dimensionality reduction.

In each of these cases, we can accomplish knowledge transfer via parameter transfer (e.g. weight updates), representation transfer (e.g. representing features differently in a target domain), or instance transfer.

Significance and Applications

Transfer learning is specifically very useful in real world setups, because of three primary reasons:

I. A classifier is trained to represent data that is defined over a specific domain and needs applied to a different domain.
II. A classifier is trained for data that quickly becomes outdated and it is too costly to continue to relabel training data.
III. A problem exists where labeling data is very expensive and transferring knowledge from a general domain to the specific domain is more cost effective.

Two large areas of application for transferring learning are currently sentiment analysis and medical image processing. Sentiment analysis is a common problem needing domain transferring simply so a sentiment algorithm for something like movie reviews could work across genre. Medical imaging requires very expensive and time consuming labeling. If this can be avoided through transferring general image knowledge, this allows for better and more cost efficient models. In general, transfer learning has shown great potential for the field of image processing, since it allows data scientists to capitalize on the hidden representations characteristic of convolutional neural networks with minimal loss in accuracy, and without the overhead computing power necessary to compute such models.

Data

The four datasets that we used for this project are the Multi-Domain Sentiment Dataset [1], 20 Newsgroups [2], and the two ECML/PKDD [3] datasets.

The **Multi-Domain Sentiment Dataset** is a dataset for sentiment analysis of reviews containing labeled examples of positive and negative Amazon reviews of items in four different domains, being Kitchen, DVD, Books, and Electronics. Each domain contains a collection of 2000 positive and negative reviews, split evenly between positive and negative, and an additional collection of some number of labeled reviews created for the purposes of additional evaluation. The features for each of these reviews are correlated words and their frequency of occurrence. This dataset is used for the sentiment classification problem, where the objective is to be able to classify a review in a domain as positive or negative.

The **20 Newsgroups** dataset was a constructed dataset stemming from a collection of 20,000 newsgroup documents split into different areas. Originally this dataset was used for text classification and text clustering however it is used in this project as a domain identification dataset, containing data from the Computer topic and the Science topic. The source domain, being the training set, is the contents of the comp.graphics, comp.os.ms-windows.misc, comp.sys.ibm.pc.hardware, comp.sys.mac.hardware newsgroups labeled positively and sci.crypt, sci.electronics, sci.med labeled negatively. The test domain is comp.windows.x and sci.space with the same labels. All of these datasets are labeled word frequency pairs. The objective of this constructed dataset is then to be able to identify what high level domain the example belongs to, be it the computer topic or the science topic.

The **ECML/PKDD** datasets are two spam classification datasets, task a being for a single source domain containing 4000 examples and task b being for multi-source domain with 15 source domains, each having 100 examples. Each of the domains is a collection of emails sent to a single different inbox, with examples being word frequency pairs labeled as either spam or not spam. These datasets are used for the spam classification problem, where given a sample email in a domain the objective is to classify if it is spam or not.

Comparison of Transfer Algorithms

Baselines:

Dataset	Baseline (Source)	Baseline (Target)	Baseline (All)
Multi-Domain Sentiment	0.774 ± 0.002	0.751 ± 0.02	0.798 ± 0.005
Newsgroup	0.840 ± 0.004	0.888 ± 0.02	0.883 ± 0.001
ECML/PKDD Task A	0.945 ± 0.0008	0.908 ± 0.02	**0.959 ± 0.004**
ECML/PKDD Task B	0.904 ± 0.004	0.930 ± 0.02	0.932 ± 0.01

Our Results:

Dataset	SCL	FEDA	STC	TrAdaboost	LWE	pLWE
Multi-Domain Sentiment	0.732 ± 0.03	**0.804 ± 0.01**	0.48 ± 0.00	0.772 ± 0.154	0.774 ± 0.0131	0.767 ± 0.0154
Newsgroup	0.751 ± 0.01	0.897 ± 0.01	0.6 ± 0.025	0.762 ± 0.237	**0.955 ± 0.00153**	0.951 ± 0.00379
ECML/ PKDD Task A	0.912 ± 0.02	0.946 ± 0.01	0.48 ± 0.043	0.844 ± 0.201	0.907 ± 0.0132	0.897 ± 0.0144

ECML/ PKDD Task B		0.944 ±0.02		0.862 ± 0.023	**0.975 ± 0.0131**	0.928 ± 0.0345

Analysis of Transfer Algorithm Comparisons

Based on the results in the table above, we performed t-tests with a 95% confidence interval between the best classifier and the second best classifier, between the second best and the third best, and so on, skipping comparisons between the baselines themselves.

Each different classifier established a mean accuracy and standard deviation over some number of iterations or cross validation folds during training.

- Baselines: 20 iterations
- FEDA: 20 iterations
- SCL: 5 iterations
- STC: 10 iterations
- TrAdaboost: 5 iterations
- LWE: 5 iterations
- pLWE: 5 iterations

For each problem, utilizing the sample size, mean, and standard deviation, we can establish an order of superiority for each learning algorithm.

Multi-Domain Sentiment

The interval between FEDA and the Baseline (ALL) classifiers was 0.00094 to 0.01106, indicating a statistically significant increase in accuracy when using the FEDA algorithm. Between the Baseline (ALL) classifier and the LWE classifier, the interval was 0.016650 to 0.031350, indicating that the Baseline classifier was statistically better than the LWE classifier. Between LWE and Baseline (Source), no t-test was needed as the means were the exact same. Between LWE and TrAdaboost, the interval was -0.157390 to 0.161390, indicating no statistical difference. Between TrAdaboost and pLWE, the interval was -0.154599 to 0.164599, again indicating no statistical difference. Between pLWE and Baseline(Target) the interval was -0.003927 to 0.035927, showing no statistical difference. Between pLWE and SCL, the interval was 0.000270 to 0.069730, indicating that pLWE showed a statistically significant increase in accuracy. Finally, between SCL and STC, a statistically significant difference in accuracy is clearly visible due to the large gap in accuracies.

Newsgroups

For newsgroups, the ranking of classifiers based on the resulting accuracy is LWE, pLWE, FEDA, Baseline (Target), Baseline (All), Baseline (Source), TrAdaBoost, SCL, and STC. There was no statistical significance between LWE and pLWE. There was a significant difference between pLWE and FEDA. For both comparisons between FEDA vs Baseline (Target) and Baseline (Target) vs Baseline (All) there was not a significant difference. There was significance between Baseline (All) and Baseline (Source). There was no significance between Baseline (Soure) vs TrAdaBoost and TrAdaBoost vs SCL. Finally, for SCL vs STC, there was a statistical significance.

ECML/PKDD Task A

FEDA was overwhelmingly the best algorithm for Task A classification, with a 95% confidence of the true mean difference between SCL lying in the range 0.0212 to 0.0468. However, SCL's gain in accuracy as compared to LWE was not statistically significant, with 95% confidence of the true mean difference falling in the interval -0.01015 to 0.03015. Although LWE consistently appears to outperform pLWE, this difference is not statistically significant, with a true mean difference of 0 appearing in the confidence interval -0.010146 to 0.030146. In a similar fashion, pLWE's true mean difference to TrAdaboost falls in the interval -0.15482 to 0.26082 with 95% confidence. Finally, the difference between TrAdaboost and STC, was extremely significantly significant, with the true mean in the interval 0.2254 to 0.5026 with 95% confidence. Consequentially, all other algorithms are also statistically superior to STC as demonstrated by this ordering.

ECML/PKDD Task B

The difference between the best classifier, LWE, and the second best classifier, FEDA, for Task B dataset was significant, as 0 does not fall in the interval containing their estimated true mean difference, 0.011367 to 0.050633 with 95% confidence. The difference between the second and third best classifiers, FEDA and Baseline (All) was also significant, as the difference in their intervals was from 0.02212 to 0.00188. Between Baseline (All) and Baseline (Target), however, the difference in t tests was not significant, from -0.00812 to 0.01212. Between Baseline (Target) and pLWE, there was also no significance, as the interval was from -0.021978 to 0.025978. For pLWE versus Baseline (Source), there was a significant difference, as the interval was 0.008651 to 0.039349. Finally, for Baseline (Source) and tradaboost, there was a significant difference, with an interval of 0.03139 to 0.05261.

Conclusions

For the Multi-Domain Sentiment dataset we found that FEDA had the highest accuracy in a significant way. This is likely due to how FEDA does well for domains where there is enough of a domain overlap to be able to generalize some of the words but not a significant enough overlap for there to be significant crossover between the weights learned for the general case as

well as the target specific case, leading to noise in the weights and worse performance. For the same reason FEDA was not the greatest classification algorithm for Newsgroup and Spam as there is significant overlap between the domains, where there is not a notable difference in what is spam for one inbox vs spam for another inbox, or what is a computer newsgroup vs what is a science newsgroup. Being a transductive algorithm, FEDA was also able to exploit the totality of the target examples in addition to the source examples.

LWE and pLWE excelled in areas where FEDA fell short, particularly where clustering on training domains was of high purity, such that its structure could be retained and applied to the testing domain, where it cannot rely on labeled data. Since LWE exploits local structural similarities between training and testing domains, it benefits from ensembling multiple models because it calculates model voting weights for each example in the testing domain. This is easily observed in ECML/PKDD Task B, which had 15 sets of spam emails assigned with word frequency pairs. Since models representing the training sets had significant structural similarity which could be exploited by the locally weighted ensemble at each example on the testing domain. This general logic holds for the 20 newsgroups dataset as well, where the same structural properties hold between the newsgroup categories. It is worth noting that pLWE consistently underperforms in comparison to LWE, since the algorithm inherently skips the step of LWE which estimates labels which are weakly predicted by the model ensemble using the average labels of more strongly predicted examples clustered around the weak example.

TrAdaBoost performed best on ECML/PKDD Task A data set followed by Multi-Domain Sentiment. For both of these problems, TrAdaBoost outperformed standard AdaBoost. This algorithm performed better than AdaBoost primarily because the source and target distribution are significantly different. TrAdaBoost performs better when leveraging split domain distributions. As seen in the performance of TrAdaBoost above, this classifier does not work well on Newsgroups because the distributions of the source and target are so close to overlapping. The extension of Multisource TrAdaBoost helped push the overall accuracy up on Amazon Sentiment and on Task B Spam.

SCL performed better on the Multi-Domain Sentiment classification problem than on the 20-Newsgroup set and the ECML/PKDD Task A datasets. This may be due to the fact that there was a high degree of domain similarity in the 20-Newsgroup and ECML sets. Due to the way SCL works, this means that the pivot features in both the source and target domains would be highly correlated with the class labels, and that by removing them and only looking at the remaining features, although they have been adapted to predict the appearance of the pivot features, there is still likely to be a loss of information. In this manner, it is expected that SCL would perform worse on prediction tasks between domains that are more similar. In comparison to the rest of the algorithms, SCL had a relatively lower accuracy rates, even compared to the baseline classifiers. This may be due to the fact that SCL is a relatively older algorithm compared to the others, and there have been numerous improvements in both technique and model choice since that paper was published.

STC may perform significantly worse than the other classifiers because it is the only unsupervised transfer learning algorithm in this comparison, which means that its performance is

likely to be lower on classification problems with labeled data. The other algorithms were all either inductive or transductive, whereas STC does not work with labeled data. As a result, it will likely not outperform a supervised learning algorithm that is able to train using labeled datasets.

References

[1] "Multi-Domain Sentiment Dataset (version 2.0)," *Multi-Domain Sentiment Dataset*. [Online]. Available: https://www.cs.jhu.edu/~mdredze/datasets/sentiment/.

[2] "20 Newsgroups," *Home Page for 20 Newsgroups Data Set*. [Online]. Available: http://qwone.com/~jason/20Newsgroups/.

[3] "Discovery Challenge," *ECML/PKDD Discovery Challenge 2006*. [Online]. Available: http://www.ecmlpkdd2006.org/challenge.html#download.

[4] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big Data*, vol. 3, no. 1, 2016.