UNIVERSITY OF MASSACHUSETTS AMHERST

CS 689 Project: Domain Adaptation for Sentiment Analysis

Aishwarya Kamath, Dung Thai, Xiang Li

December 19, 2016

1. INTRODUCTION

Domain adaptation aims to generalize a classifier that is trained on a source domain, for which typically plenty of training data is available, to a target domain, for which labeled data is scarce[3]. Further, the classifier uses the unlabelled target data along with the source data.[4] Good generalisation across domains is essential in several application areas of machine learning, where the data is imbalanced as stated above. Further, it is often the case that data in the source and target domains themselves have different radically different distributions. This presents a major obstacle in adapting predictive models.

Supervised machine learning methods only perform well when the given extensive labeled data are from the same distribution as the test distribution. Test error of supervised methods generally increases in proportion to the difference between the distributions of training and test examples[7].

2. METHODS

2.1. MARGINALISED STACKED DENOISING AUTO-ENCODERS

Glorot et al. [5] proposed a deep learning approach which learns to extract a meaningful representation for data in an unsupervised fashion. They basically train a stacked denoising auto-encoder which tries to reconstruct the input vector optimizing the distance between the random corrupted input and actual input data. The whole framework in based on the work of vincent 2008 where they stated denoising auto-encoder.[9]

An auto-encoder is comprised of an encoder function h(Âů) and a decoder function g(Âů), with typically the dimension of encoder function smaller than that of its argument. The reconstruction of input x i.e output from auto-encoder is given by $r(x) = g(h(x))$. When training an auto-encoder, one typically trains it to minimize a form of reconstruction error $loss(x, r(x))$. A Denoising Auto-Encoder [9] refers to the case in which the input vector x is randomly corrupted into a vector $\tilde{x}$ using dropout or random gaussian noise, and the model is trained to denoise, i.e., to minimize a denoising reconstruction error $loss(x, r(\tilde{x}))$.

The authors from Glorot et al.[5] trained a stacked denoising auto-encoder(SDA) to reconstruct the input vectors (without the labels) on the union of the source and target data. They stacked the denoisers into deep learning architectures and learnt the hidden representation in a greedy fashion and use the output from t^{th} layer as the input of $t+1^{th}$ layer. Finally, a linear classifier is trained on the transformed labeled data of the source domain and test on both target and source domains. Support Vector Machines (SVM) being known to perform well on sentiment classification [10] were used for this purpose. They showed that SDAs are able to disentangle hidden factors which explain the variations in the input data, and automatically group features in accordance with their relatedness to these factors [5]. This helps transfer across domains as these generic concepts are invariant to domain-specific vocabularies.

Although these stacked denoising auto-encoders showed promising results, SDAs are limited by their high computational cost. They are significantly slower to train than competing algorithms, primarily because of their reliance on iterative and numerical optimization to learn model parameters. The challenge is further compounded by the dimensionality of the input data and the need for computationally intensive model selection procedures to tune hyper parameters. Consequently, even a highly optimized implementation would take days to train a SDA. Hence, we decided to use a faster and much easier to use algorithm called Marginalised Stacked Denoising Auto-encoders(mSDA)[3].

Marginalized Denoising Auto-encoders(mSDA) adopts the greedy layer-by-layer training of SDAs. However, a crucial difference is that they use linear denoisers as the basic building blocks. Under this setting, the random feature corruption can be marginalized out. In this method, linear denoisers are used as building blocks so that it is possible to marginalise out the random feature corruption, ensuring that the entire optimisation is convex and can be solved in a closed loop manner. The algorithm can be written in ten lines in MATLAB as showing in Algorithm1. But we implemented our own python version.

Algorithm 1 mSDA for learning hidden representation

```
1: function [WS,HS]=MSDA($X, p, l$)
2:     $[d, n] = size(X)$;
3:     $Ws = zeros(d, d+1, 1)$;
4:     $hs = zeros(d, n, 1+1)$;
5:     $hs(:,:,1) = X$;
6:     for $t = 1 : l$ do
7:         $[Ws(:,:,t), hs(:,:,t+1)) = mSDA(hs(:,:,t), p)]$;
```

2.2. Correlation Alignment

The work of Sun et al[8] describes a "frustratingly easy" method for aligning the source and target distributions. CORAL(Correlation Alignment) aligns the input feature distributions of the source and target domains by exploring their second-order statistics. To explain more clearly, this means they propose to re colour the whitened source features with the target distribution's covariance matrix. As we have seen after conducting experiments using this method, this procedure works well only if the source and target distributions are sufficiently different. The advantage of using this method is that even for tasks such as object recognition, it requires only to calculate the source and target covariances as opposed to recent deep CNN adaptation approaches[6].

Algorithm 2 CORAL for Unsupervised Domain Adaptation

```
1: $C_S = cov(D_S) + eye(size(D_S, 2))$
2: $C_T = cov(D_T) + eye(size(D_T, 2))$
3: $D_S = D_S * C_S^{(-1/2)}$                ▷ Whitening source
4: $D_S^* = D_S * C_S^{(1/2)}$               ▷ Re-colouring with target covariance
```

This algorithm can be applied to any classifier because it changes only the features. We know already that input feature normalisation improve learning in many machine learning methods. However, CORAL goes an extra step by aligning the source and target distributions. So this addresses the case where although the features are normalized to have zero mean and unit variance in each dimension, the differences in correlations present in the source and target domains cause the distributions to be different.

2.3. Combining the two

We wish to see the result of combining the above mentioned methods. We propose to do this, by extracting the MSDA features in an unsupervised manner and using those as inputs to the CORAL algorithm. The classifier that is trained on the resultant features should provide superior accuracy. We believe the two algorithms optimize different objectives so they will be complementary and produce better results. This comes from our understanding that mSDA tries to disentangle features in a lower manifold and coral capitalises on the difference in the distributions of the source and target. So if the mSDA manages to give hidden representation features which increase the difference in the distributions of the source and target domains, CORAL will improve the classification further giving better results than either of the two algorithms.

3. Dataset

The data set we are using is the reduced Amazon reviews dataset. It contains reviews of products on Amazon for 4 domains - Books, Electronics, Kitchen and DVD. Each of the domains have two classes for the sentiment - either positive or negative. We have two versions of this dataset: The first being extracted by us from the original Amazon reviews dataset according to the data processing described in the work of [4] and [3]. The second is the dataset provided by the authors of [2]. We report the results using both datasets and our analysis for each.

4. Experiments

Initially we extracted the features ourselves, from the reviews data available online[1] and constructed a bag of words representation. Both the papers state that they perform the algorithms on standard bag of words features. However, we saw a large difference in the accuracies for the in domain as well as cross domain tasks on the Amazon reviews dataset, when applied to this standard bag of words features as opposed to the pre-processed data that was explained in the works of Hana Ajakan et al on Domain Adversarial Neural Networks [2]. This pre processing is described as follows: Each review text is treated as a bag of words and transformed into binary vectors encoding the presence/absence of unigrams and bigrams. Hence, we first tabulate our results on the bag of words features as well as on the pre-processed data and run some data visualisations to understand why this happens.

We ran the CORAL algorithm for all the cross domain cases and compared them with the no adaptation case (NA), which corresponds to training the classifier on the source domain and testing on the target domain.

CORAL appears to do better than the no adaptation cases only in some cases. So we decided to perform some data visualisation so as to better understand what is happening.

In we have calculated the t-SNE for the books domain bag of words features with no further processing and we see from Figure 4.1 that there is no clear clustering which would allow any classifier to be able to classify the two sentiments well. t-SNE is a dimensionality reduction technique that maintains the small-scale structure (it describes which data points are close to which other) of the space, so it is useful as a method for visualising the separability of the data. Some other dimensionality reduction methods like PCA do not separate out the data as the dimensions disappear, instead such methods tend to leave the data overlapping. This makes it difficult to deduce any clear understanding about the separability of the data when it is in higher dimensions.

More concretely, A t-SNE plot with no clear separable cluster like structures, will usually translate to a classifier not being able to do very well on the data. On the other hand, clearly separated data in the t-SNE plot implies that in a higher dimension, the data has enough variance to give good results during classification.

On the other hand, the t-SNE plot of the pre-processed data is shown below in Fig 4.2 and we can clearly see how separable the data is.

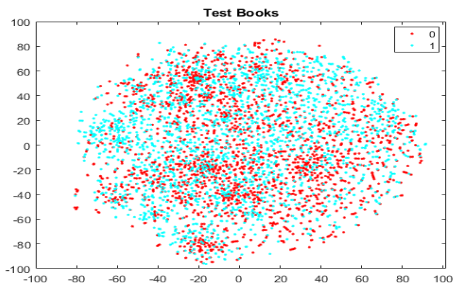

Figure 4.1: t-SNE Raw BOW

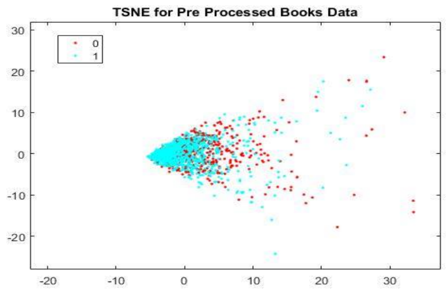

Figure 4.2: t-SNE Pre-Processed

On the other hand, Fig 4.3 shows a t-SNE plot of the image dataset where CORAL shows huge gains [4], as compared to text domains. Here we can clearly see the 10 clusters of the different classes.

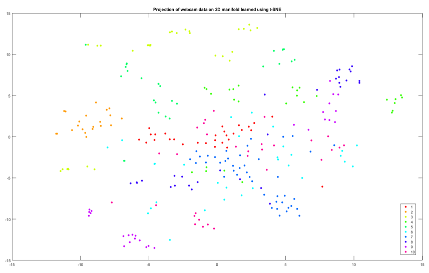

Figure 4.3: t-SNE Webcam (OFFICE-10 dataset)

We observed that the CORAL algorithm shows significant improvement over the "No Adaptation" case only when the distributions are sufficiently different as shown clearly by the plots of the CDF of Kitchen, Electronics and of the DSLR, Webcam datasets shown below in Fig 4.4 and Fig 4.5 respectively.

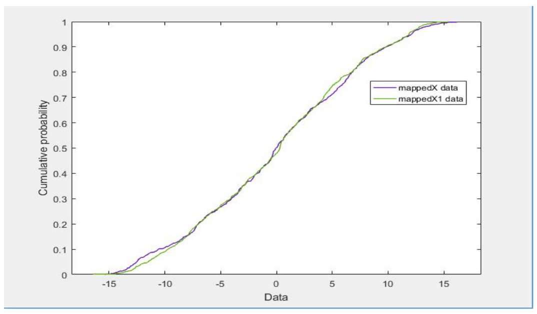

Figure 4.4: CDFs Kitchen, Electronics

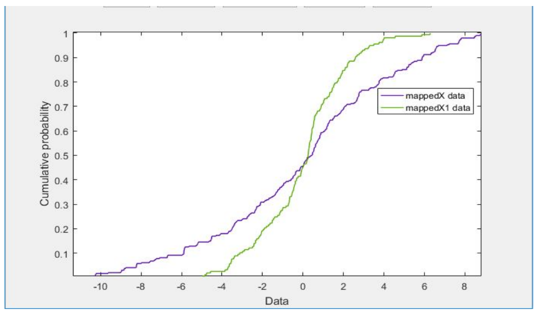

Figure 4.5: CDFs Webcam, DSLR

Now coming to within the text domain, we notice that CORAL does much better on the Books to Kitchen shift as compared to the Kitchen to Electonics shift, again, due to the difference in their distributions as shown in Fig 4.6 and Fig 4.7.

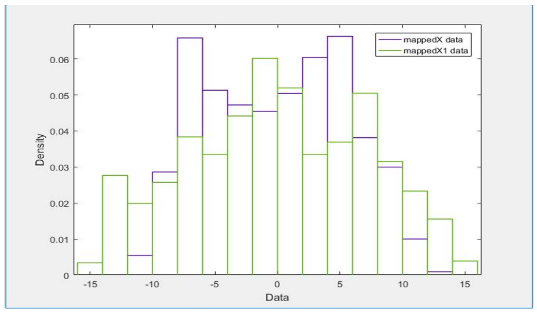

Figure 4.6: PDFs Books, Kitchen

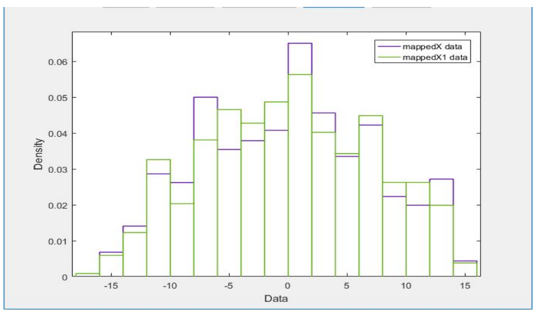

Figure 4.7: PDFs Kitchen, Electronics

Finally by running mSDA on the pre processed data, we checked whether there is any difference in the distributions of the domains before and after applying the algorithm. The following figure confirmed out belief in the fact that mSDA features passed as input to CORAL algorithm should produce better results.

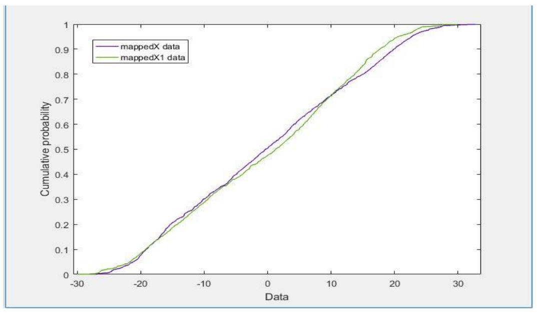

Figure 4.8: CDFs Books, Kitchen with pre-processed data

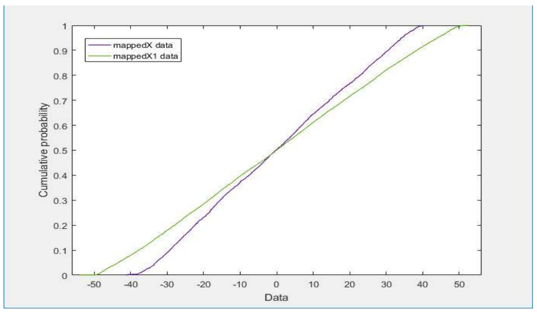

Figure 4.9: CDFs Books, Kitchen with mSDA hidden representations

5. Results

The accuracies for each of the domains are listed below.

Table 5.1: Self processed data

Model	BOW-SVM	BOW - CORAL	BOW - mSDA	TF-IDF - mSDA	Embeddings - mSDA
B=>D	50.38	50.48	45.1	43.6	**58.7**
B=>E	48.67	49.11	44.8	40.8	**52.7**
B=>K	52.00	53.12	52.3	47.1	**57.2**
D=>B	47.36	52.96	46.5	49.2	**57.5**
D=>E	48.84	50.12	44.8	48.0	**55.6**
D=>K	48.43	51.02	50.6	49.9	**55.4**
E=>B	47.82	47.84	53.0	50.1	**56.1**
E=>D	51.04	51.07	50.4	53.7	**58.0**
E=>K	50.19	50.85	53.2	58.0	**60.9**
K=>B	50.36	50.56	52.7	48.4	**54.9**
K=>D	52.47	52.47	53.4	49.8	**58.3**
K=>E	47.51	47.90	54.0	50.2	**61.9**

Table 5.2: Pre-processed data

Model	SVM Baseline	CORAL	mSDA	DANN[2]	coral + msda
B=>D	78.4	79.58	76.5	82.9	**82.16**
B=>E	74.3	74.79	73.2	80.4	**74.80**
B=>K	74.6	77.30	77.2	84.3	**83.41**
D=>B	71.6	77.61	77.4	82.5	**82.07**
D=>E	72.1	75.85	69.7	80.9	**76.95**
D=>K	74.0	79.50	68.3	84.9	**79.95**
E=>B	70.0	71.67	73.2	77.4	**77.31**
E=>D	70.0	74.84	74.7	78.1	**78.34**
E=>K	83.6	84.15	82.1	88.1	**85.28**
K=>B	69.2	70.05	70.6	71.8	**77.31**
K=>D	71.9	73.92	73.7	78.9	**79.00**
K=>E	83.3	83.38	82.5	85.6	**85.21**

As we can see from all the tables, using pre-processed data gives us a huge gain even for the SVM baseline(from 50% to 70%). Another interesting observation is that using embeddings+ raw BOW improves the accuracy over using just raw BOW. Further, using the pre processed data works even better than our implementation using embeddings on BOW. Our understanding of this is that using word features that cut across domains improves results as opposed to our system where we are using off-the-shelf embeddings for this task. The paper [4] cites pre-processed data that takes into account only data from all the domains while performing the embedding and is hence more task focused as opposed to our method of using off the self embeddings.

Using mSDA gives us a gain in most cases. However, due to computational restrictions we have not been able to use a 5 layer model and have hence reported results using 4 layers with a corruption probability of 0.8(a hyper parameter for our model). These results however, are also comparable to those stated in the paper. We go further by using these 4 layer outputs to be used as input to the CORAL algorithm. This gave us results better than either of the two methods separately and are even comparable to the state of the art Domain Adversarial Neural Network method. DANN result is reported directly from the paper [2].

6. Conclusion

- Data pre-processing is very important.
- CORAL results depend heavily on the source and target data distribution. If the source and target distributions are different, CORAL usually can give us a gain.
- mSDA on the other hand works better when the distributions are similar.
- By combining the two algorithms we have shown that we can obtain superior results than either of the two methods separately. We believe if we have the fully trained mSDA features, combining mSDA and CORAL will definitely beat the state of the art results.

7. Future work

We wish to extend this work by firstly using an mSDA model with 5 layers and further tune the hyper parameter for noise level to get better hidden representations. These will be used to obtain a better feature by using the CORAL algorithm and then passed as input to a Domain Adversarial Neural Network. We expect superior results using this model.

References

[1] Multi-domain sentiment dataset.

[2] Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, and Mario Marchand. Domain-adversarial neural networks. *arXiv preprint arXiv:1412.4446*, 2014.

[3] Minmin Chen, Zhixiang Xu, Kilian Weinberger, and Fei Sha. Marginalized denoising autoencoders for domain adaptation. *arXiv preprint arXiv:1206.4683*, 2012.

[4] Hal Daumé III. Frustratingly easy domain adaptation. *arXiv preprint arXiv:0907.1815*, 2009.

[5] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 513–520, 2011.

[6] Mingsheng Long and Jianmin Wang. Learning transferable features with deep adaptation networks. *CoRR, abs/1502.02791*, 1:2, 2015.

[7] Baochen Sun, Jiashi Feng, and Kate Saenko. Return of frustratingly easy domain adaptation. *arXiv preprint arXiv:1511.05547*, 2015.

[8] Baochen Sun, Jiashi Feng, and Kate Saenko. Return of frustratingly easy domain adaptation. *arXiv preprint arXiv:1511.05547*, 2015.

[9] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[10] Suchita V Wawre and Sachin N Deshmukh. Sentiment classification using machine learning techniques.

A. mSDA (SENTIMENT.PY)

```
import numpy as np
from sklearn import svm
from scipy import io

from prepro import *
from msda import *

num_layers = 4
noises = [0.5, 0.6, 0.7, 0.8, 0.9]

domains = ['books', 'dvd', 'electronics', 'kitchen']
X_train, y_train, train_offset = [], [], [0]
X_test, y_test, test_offset = [], [], [0]
use_embeddings = False
use_tf_idf = False

for domain in domains:
        train_path = "data/{0}_train.svmlight".format(domain)
        test_path  = "data/{0}_test.svmlight".format(domain)

        X, y = create_input(train_path, use_embeddings=True)
        X_train.append(X)
        y_train.extend(y)
        train_offset.append(train_offset[-1] + len(X))

        X, y = create_input(test_path, use_embeddings=True)
        X_test.append(X)
        y_test.extend(y)
        test_offset.append(test_offset[-1] + len(X))

X_train = np.vstack(X_train)
X_test = np.vstack(X_test)

noise = noises[3]
X_train_reps, train_mappings = mSDA(X_train.T,
        noise, num_layers, verbose=True)
X_test_reps = mSDA_forward(X_test.T, train_mappings)


for i, domain in enumerate(domains):
        b, e = train_offset[i], train_offset[i+1]
        classifier = svm.SVC(kernel = 'linear')
        .fit(X_train_reps[b:e], y_train[b:e])
        train_save = {}
        train_save['train_x'] = X_train_reps[b:e]
        train_save['train_y'] = y_train[b:e]
        io.savemat(domain+"train", train_save)

        b, e = test_offset[i], test_offset[i+1]
        preds = classifier.predict(X_test_reps[b:e])
        acc = np.mean(preds == y_test[b:e])
        print "Accuracy␣on␣%s␣domain" % domain, acc

        for j, tdomain in enumerate(domains):
                if i == j:
                        continue
                b, e = test_offset[j], test_offset[j+1]
                preds = classifier.predict(X_test_reps[b:e])
                acc = np.mean(preds == y_test[b:e])
                test_save = {}
                test_save['test_x'] = X_test_reps[b:e]
                test_save['test_y'] = y_test[b:e]
                io.savemat(domain+"To"+tdomain+"test", test_save)
                print "Adapt␣to␣%s␣domain" % tdomain, acc
        print ""
```

B. mSDA (PREPRO.PY)

```
import numpy as np
from sklearn.datasets import load_svmlight_files
from scipy import io

def create_input(dataset_fn, use_embeddings=False, word_mappings=None):
    if 'train' in dataset_fn:
        label = 'train'
    else:
        label = 'test'

    if use_embeddings:
        X, y = load_svmlight_files([dataset_fn])
        X = np.array(X.todense())
        y = np.array((y + 1) / 2, dtype=int)
        return X, y

    return X, y

if __name__=='__main__':
    load_amazon('dvd', 'electronics', './data/' , verbose=True)
```

C. mSDA (msda.py)

```
import numpy as np

def mDA(X, noise, eta):
    """
    inputs:
        X : d x n input (Transpose of the usual data-matrix)
        noise: corruption level
        eta: regularization

    outputs:
        hx: d x n hidden representation
        W: d x (d+1) mapping
    """
    d, n = np.shape(X)

    # adding bias
    Xb = np.vstack((X, np.ones(n)))

    # scatter matrix S
    S = np.dot(Xb, Xb.T)

    # corruption vector
    q = np.ones((d+1, 1)) * (1.-noise)
    q[-1] = 1

    # Q: (d+1)x(d+1)
    Q = S*np.dot(q,q.T)
    Q[np.diag_indices_from(Q)] = q.T[0] * np.diag(S)

    #P: dx(d+1)
    P = S[0:-1,:] * q.T

    # final W = P*Q^-1, dx(d+1)
    reg = eta * np.eye(d+1)
    reg[-1,-1] = 0
    W = np.linalg.solve( Q.T+reg, P.T ).T

    hx = np.tanh( np.dot(W, Xb) )
    return hx, W


def mSDA(X, noise, nb_layers, verbose=False):
    """
    inputs:
        X : d x n input Transpose of the usual data-matrix)
        noise: corruption level
        nb_layers: number of layers to stack
    outputs:
        allhx: (1+nb_layers)*d x n stacked hidden representations
        W_list: list of mapping (of size nb_layers)
    """
    eta = 1e-05
    allhx = X.copy()
    prevhx = X
    W_list = []

    for i in range(nb_layers):
        if verbose: print('layer␣=', i)
        newhx, W = mDA(prevhx, noise, eta)
        W_list.append(W)
        allhx = np.vstack((allhx, newhx))
        prevhx = newhx

    return allhx.T, W_list


def mSDA_forward(X, W_list):
    """
    inputs:
        X : d x n input (Transpose of the usual data-matrix)
        noise: corruption level
        W_list: list of mapping (of size nb_layers) learned by mSDA.

    outputs:
        allhx: (1+nb_layers)*d x n stacked hidden representations of X.

    """
    _, n = np.shape(X)
    hx = X
    # print X.shape

    allhx = X.copy()
    for W in W_list:
        hxb = np.vstack(( hx, np.ones(n)) )
        # print W.shape
        # print hxb.shape
        hx = np.tanh( np.dot(W, hxb) )
        allhx = np.vstack( (allhx, hx) )

    return allhx.T
```

D. mSDA (baseline.py)

```
import numpy as np
from sklearn import svm
from scipy import io

from prepro import *
from msda import *

domains = ['books', 'dvd', 'electronics', 'kitchen']
X_train, y_train, train_offset = [], [], [0]
X_test, y_test, test_offset = [], [], [0]
use_embeddings = False
use_tf_idf = False

for domain in domains:
        train_path = "data/{0}_train.svmlight".format(domain)
        test_path  = "data/{0}_test.svmlight".format(domain)

        X, y = create_input(train_path, use_embeddings=True)
        X_train.append(X)
        y_train.extend(y)
        train_offset.append(train_offset[-1] + len(X))

        X, y = create_input(test_path, use_embeddings=True)
        X_test.append(X)
        y_test.extend(y)
        test_offset.append(test_offset[-1] + len(X))

X_train = np.vstack(X_train)
X_test = np.vstack(X_test)
X_train_reps = X_train
X_test_reps = X_test

for i, domain in enumerate(domains):
        b, e = train_offset[i], train_offset[i+1]
        classifier = svm.SVC(kernel = 'linear')
        .fit(X_train_reps[b:e], y_train[b:e])
        train_save = {}
        train_save['train_x'] = X_train_reps[b:e]
        train_save['train_y'] = y_train[b:e]
        io.savemat(domain+"train", train_save)

        b, e = test_offset[i], test_offset[i+1]
        preds = classifier.predict(X_test_reps[b:e])
        acc = np.mean(preds == y_test[b:e])
        print "Accuracy␣on␣%s␣domain" % domain, acc

        for j, tdomain in enumerate(domains):
                if i == j:
                        continue
                b, e = test_offset[j], test_offset[j+1]
                preds = classifier.predict(X_test_reps[b:e])
                acc = np.mean(preds == y_test[b:e])
                test_save = {}
                test_save['test_x'] = X_test_reps[b:e]
                test_save['test_y'] = y_test[b:e]
                io.savemat(domain+"To"+tdomain+"test", test_save)
                print "Adapt␣to␣%s␣domain" % tdomain, acc
        print ""
```