20 news text classification

EE 660 Course Project

Project Type: (1) Design a system based on real-world data

Number of student authors: 1

Jingxiang Zhang – jzhang92@usc.edu

12/1/2022

1. Abstract

In this problem, I will use 20 news text data set with supervise learning and transfer learning. I will first use word embedding method to preprocess the data. In supervise learning part, I will try different classification model to classify the data and make comparison. Pick up the best model and test it on the test set. In transfer learning part, I will separate the data again to fit for the transfer learning problem. Then try Subspace Alignment, Tr Ada Boost, and Importance Weighting combine with the supervise learning model. Finally, I will analyze the result and make conclusions.

2. Introduction

2.1. Problem Type, Statement and Goals

This is a classification problem. The dataset combined with 20 classes of news, which include computer topic, recreation topic, science topic and so forth. And my goal is to classify 20 news datasets. My work includes 2 parts, machine learning part and transfer learning part.

For part 1, I will first make a comparison between different classification models, as supervise learning. A good model that can correctly classify the news can help us build a Recommender Systems. We can label the news by the model and recommend to the user who interested in this area. The main difficult of this problem is the text data itself. Encoding the text data will make the input data become high dimensionality of feature space, and sparse in each feature. All of these will decrease the accuracy of classification.

Then in part 2, I will try different transfer learning methods combine with the stage 1 models. Not all the methods are fitful for this problem, thus I will compare the results and pick the most fitful one. This transfer learning part is also useful in some cases. For example, we have lots of machine learning news with a few of deep neuron network news since it is a new topic these years. The information gained from machine learning news can help us build a better model for the deep neuron network news by transfer learning strategy.

2.2. Literature Review

When dealing with text data, typically the first step is to do word embedding, which is to encode each word from the news into a vector. Varies of methods are put forward to do word embedding, for example, one hot is the simplest way, other method like bag of words model, N-gram model, TF-IDF model, or a deep learning model called Word2Vec are even better than one-hot coding [1]. I will only use one-hot encoding and put my effort on making comparison between different models.

2.3. My Prior and Related Work - None

2.4. Overview of My Approach

2.4.1 Main topic – Supervise Learning

The baseline systems are

- The trivial baseline: It always decide the majority class.
- The non-trivial baseline: use multiclass SVM classifier with linear kernel.

I will use multiclass SVM classifier with different kernel, Multiclass Logistic Regression, Multinomial Naive Bayes, K-Nearest Neighbor, Decision Tree, Random Forest, Ada Boost, and Multilayer Perceptron. Classify the dataset into 20 classes and compare the accuracy of each model.

2.4.2 Extension – Transfer Learning

The baseline system is train on the source domain and use it to predict the label on the target domain. When training on the source domain, the target domain data will not be used to help training.

I will use Subspace Alignment, Tr Ada Boost, and Importance Weighting combine with the previous model, and compare the accuracy of each model.

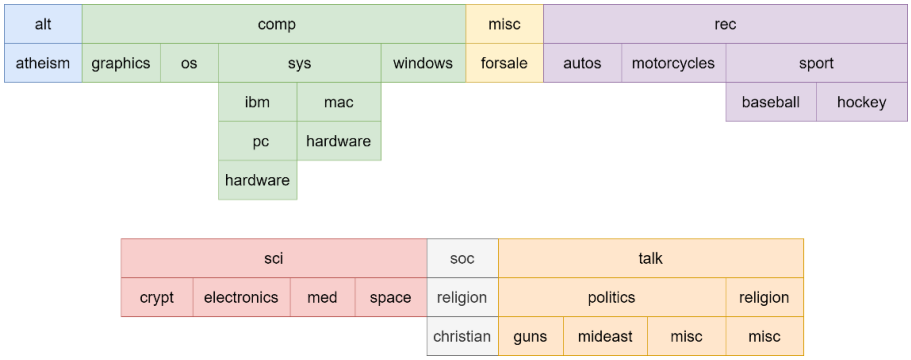
3. Implementation

You should mention which libraries and functions you used but avoid including code in your report. (For example, stating “we standardized all real-valued features, and

recast all categorical features using one-hot encoding" and also stating the functions used in your code for this, is fine

3.1. Data Set

The dataset I use is 20news dataset [2], which has several top-level categories, and for each top-level category it has many second-level categories, and even third-level categories.



From the plot above, there are 7 top-level categories, and 20 sub-level categories. For each category, it has the numbers of data as the following table show.

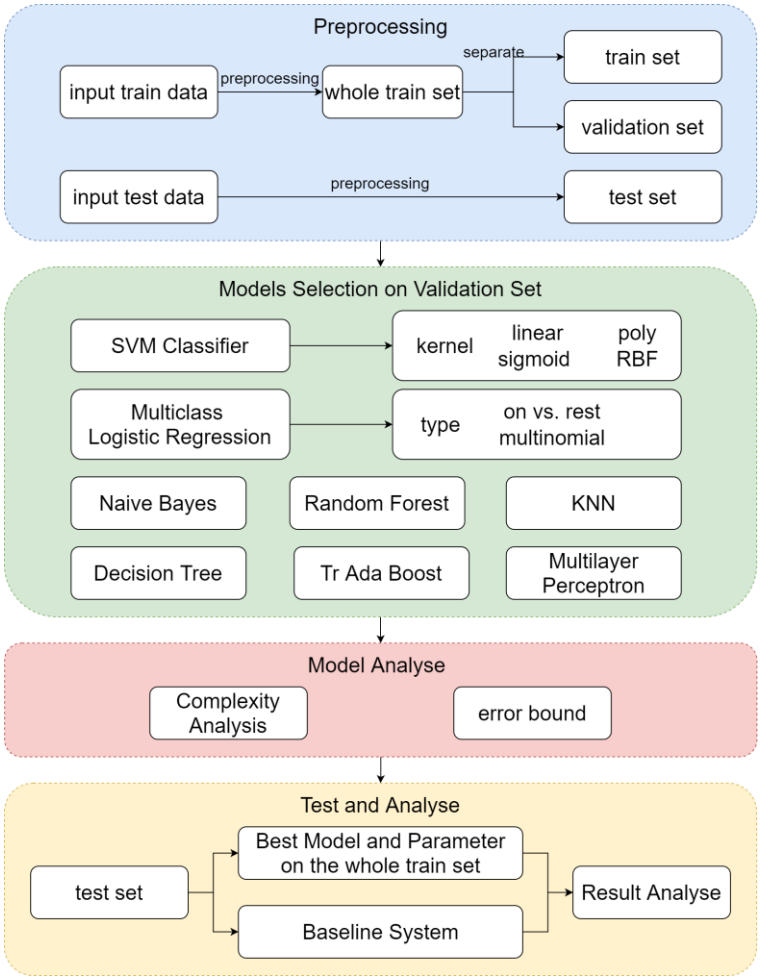
Class	1	2	3	4	5	6	7	8	9	10
Train Num	480	581	572	587	575	592	582	592	596	594
Test Num	318	389	391	392	383	390	382	395	397	397
Class	11	12	13	14	15	16	17	18	19	20
Train Num	598	594	591	594	593	599	545	564	464	373
Test Num	399	395	393	393	392	398	364	376	310	251

There are totally 11269 train data points, and 7505 test data points. The maximum variation of the dataset is 1.61. So, the dataset is balanced when only considering the sub level category.

3.2. Dataset Methodology

Before training, I will separate the training set into training set and validation set. The validation set will take 20 percent of the original training data, and the remaining will be used to train the validation model. When pick up the most fitful model, I will use the whole training set to train the model and test it on the test set.

The flowchart is listed as follows:



3.3. Preprocessing, Feature Extraction, Dimensionality Adjustment

This original dataset is text data, the typical preprocessing method for text data is word embedding. One-hot is one of the most classical method and will be used in this project. The idea of it is to transfer each word into the index of the dictionary. The dataset has already done this preprocessing for us.

One drawback is that this models only considers the statistics of the word in one article. For example, one article from rec.autos labels have the following statistic, and the context or the arrangement of those words will not be kept.

```
of:2 the:2 in:1 to:2 for:1 or:1 an:2 that:1 any:2 this:2 about:1 truth:1 friend:1 used:1 make:1 also:
1 popular:1 was:1 series:1 science:1 me:1 called:1 ago:1 believe:1 too:1 car:1 either:1 couple:1 tol
d:1 appreciate:1 thanks:2 info:2 months:1 tim:1 published:1 bmw:1 ix:1 rumor:1 awd:2 autoweek:1
```

3.4. Training Process

All of the model below is implemented by sklearn.

1. Baseline system: For trivial model, it always decides the majority class, and for non-trivial model, it will use multiclass SVM classifier with linear kernel. I choose those two models as baseline system since it is most widely used. The time complexity for trivial model for training is O(N) for given N data points, and for each query, the time complexity is O(1) since it only need to output the label. And the time complexity for non-trivial model is O(NDC) for linear kernel SVM, according to this article [3], where D is feature number, and C stand for class number, and O(DC) for each query.

2. multiclass SVM classifier: Traditional SVM can only handle two classes question. By using multiclass SVM, it uses one vs. rest strategy. That is, for K classes classification problem, build K one vs. rest SVM models. For given test data, try to classify it by all the K models, and pick the model that give the highest confidence. I will use different parameters, which include linear kernel (also is no-trivial model), sigmoid kernel, radio base function (RBF) kernel with λ = 1/features and 0.1, polynomial kernel with degree = 3. The time complexity for SVM is $O(N^2DC)$ for RBF kernel using SMO solver according to this article [3], therefore it is slow for the given dataset.

3. Multiclass Logistic Regression: The same strategy as multiclass SVM classifier, using one vs. rest. Logistic Regression is a very import model in machine learning area, and it's also the base of deep neuron network. I will try the multinomial strategy with newton-cg, lbfgs, sag, and saga solver, with maximum of 500 iterations. the time complexity is O(NDC) according to this [4].

4. Gaussian Naïve Bayes: For Gaussian Naïve Bayes model, no extra parameter is needed. The formula is: $P(y_i|x) = P(x|y_i)P(y_i)/\sum_1^m P(x|y_j)P(y_j)$. All the variance and mean parameters in the conditional probability for this formula need to be estimated by Gaussian distribution. Mind that Naïve Bayes suppose that all the features are independent with each other. But appearantly it's not in this dataset. Therefore, Naïve Bayes model may not work well. It select the class with highest probability. The time complexity for building this model is O(NDC) [5] since there are D*C of Gaussian distribution need to be estimated and estimate each one is O(N). The query complexity is O(DC).

5. K-Nearest Neighbor: This method will search for the K-Nearest training points for a given test points and output the label with highest vote. K-Nearest neighbor algorithm typically use KD tree to store training data, therefore the time complexity of training this model is the same as the time complexity of building a KD tree, which is O(NDlog(N)), and the query complexity for one point is O(Klog(N)). I will set neighbor = 3, 5, or 7 and check the result.

6. Decision Tree: I will use Gini index as criterion, tree will be divided until all the leaf is pure. Decision Tree is always easily to overfitting. And the time complexity is O(DNlog(N)) according to this article [6].

7. Random Forest: I will generate 100 trees, and for each tree, set the using features equal to all the features or square root of all the features. Each tree has the same parameters as the Decision Tree. For a Random Forest with generating M trees, the time complexity is $O(MDN^{0.5}log(N))$.

8. Ada Boost: I will use tree stump (maximum depth equal to 2) for each classifier. And for each classifier, use the same parameter setting as the Decision Tree. I will tree 300, 600, and 1000 estimators (decision tree stump). The time complexity is O(NDM).

9. Multilayer Perceptron: Multilayer perceptron model is closely related to deep learning. I will only use sklearn to implement this model. Use adam for gradient descent strategy, learning rate = 0.0001, batch size = 200, epoch = 100, use ReLU as activation function. I will use the model with no hidden layer, one hidden layer with size = 100, one hidden layer with size = 256. Because the multilayer perceptron in sklearn only use one CPU core to do the training, it is very slow, so I can't add more layers. Suppose the hidden layer have T nodes. The first matrix is DT, and the second matrix is TC. Suppose we train the model for K iteration, then the time complexity is O(KNT(D+C))).

3.5. Model Selection and Comparison of Results

Result for train set (not the whole train set). All the model that accuracy higher than 0.99 is bold.

model	accuracy	model	accuracy
trivial	0.054	KNN_3	0.716
SVM_linear	**0.999**	KNN_5	0.662
SVM_sigmoid	0.180	KNN_7	0.620
SVM_RBF_auto	0.128	DT_nfeature	**0.999**
SVM_RBF_0.1	**0.997**	DT_sqrt	**0.999**
SVM_poly_3	0.163	AdaBoost_300	0.478
MLR_OVR	**0.999**	AdaBoost_600	0.429
MLR_Multi_Newton	**0.999**	AdaBoost_1000	0.495
MLR_Multi_lbfgs	**0.999**	random_forest	**0.999**
MLR_Multi_sag	0.802	MLP_no_hidden	**0.999**
MLR_Multi_saga	0.741	MLP_100	**0.999**
Multinomial_Naive_Bayes	0.940	MLP_256	**0.999**

Result for validation set. All the model that accuracy higher than 0.8 is bold.

model	accuracy	model	accuracy
trivial	0.050	KNN_3	0.423
SVM_linear	0.728	KNN_5	0.407
SVM_sigmoid	0.158	KNN_7	0.395
SVM_RBF_auto	0.121	DT_nfeature	0.588
SVM_RBF_0.1	0.081	DT_sqrt	0.411
SVM_poly_3	0.117	AdaBoost_300	0.438
MLR_OVR	**0.843**	AdaBoost_600	0.370
MLR_Multi_Newton	**0.828**	AdaBoost_1000	0.394
MLR_Multi_lbfgs	**0.828**	random_forest	**0.802**
MLR_Multi_sag	0.716	MLP_no_hidden	**0.863**
MLR_Multi_saga	0.676	MLP_100	**0.881**
Multinomial_Naive_Bayes	**0.847**	MLP_256	**0.866**

I pick up all the models that accuracy greater than 0.8 on the validation set, and calculate the accuracy on each class, and here is what I got. From the figure below, we can find that all of the model are not good at classify comp.os.ms-windows.misc, comp.sys.ibm.pc.hardware, comp.windows.x, and sci.electronics.

model	athe	graph	misc	pc	mac	win	forsale	autos	motor	baseball	hockey	crypt	elec	med	space	christ	guns	mideast	politics	religion	sum
MLR_OVR	0.902	0.804	0.752	0.610	0.833	0.781	0.853	0.917	0.889	0.920	0.885	0.921	0.761	0.899	0.898	0.928	0.832	0.935	0.837	0.625	0.843
MLR_Multi_Newton	0.902	0.795	0.752	0.619	0.778	0.763	0.872	0.883	0.874	0.911	0.858	0.889	0.717	0.916	0.881	0.912	0.815	0.925	0.802	0.611	0.828
MLR_Multi_lbfgs	0.902	0.795	0.752	0.619	0.778	0.763	0.872	0.883	0.874	0.911	0.858	0.889	0.717	0.916	0.881	0.912	0.815	0.925	0.802	0.611	0.828
Multinomial_Naive_Bayes	0.882	0.884	0.616	0.737	0.870	0.763	0.569	0.892	0.926	0.955	0.956	0.968	0.761	0.941	0.932	0.952	0.891	0.953	0.884	0.444	0.847
random_forest	0.784	0.741	0.792	0.610	0.815	0.763	0.881	0.825	0.867	0.857	0.929	0.937	0.593	0.815	0.898	0.936	0.866	0.925	0.698	0.264	0.802
MLP_no_hidden	0.931	0.866	0.792	0.644	0.843	0.772	0.862	0.900	0.889	0.938	0.920	0.921	0.761	0.916	0.881	0.944	0.882	0.953	0.884	0.722	0.863
MLP_100	0.931	0.857	0.776	0.754	0.861	0.807	0.844	0.958	0.904	0.920	0.947	0.952	0.743	0.908	0.932	0.952	0.916	0.963	0.907	0.722	0.881
MLP_256	0.931	0.857	0.816	0.653	0.870	0.781	0.807	0.942	0.889	0.920	0.938	0.937	0.805	0.908	0.873	0.904	0.891	0.944	0.860	0.736	0.866
average_accuracy	0.896	0.825	0.756	0.656	0.831	0.774	0.820	0.900	0.889	0.916	0.912	0.927	0.732	0.902	0.897	0.930	0.863	0.940	0.834	0.592	

From the training and validation accuracy table above, it shows that:

1. Multiclass Logistic Regression, Naïve Bayes, Random Forest, and Multilayer Perceptron have the best accuracy over the other model.

2. For SVM model, linear kernel get the best performance, and all the other kernels' performance are bad. That is because in high dimension feature space with sparse data points, RBF kernels is easy to overfitting, linear kernel is just fine. I'm not sure why sigmoid and polynomial kernels are not good even in the training set. I guess maybe because the curve they fit is not correct for this dataset.

3. KNN get an average performance as I expected. KNN will not work well for the noise and sparse data like this one. KNN prefer dense training set which need more data points. From the result, we can also find that the larger of the k, the worse of the result.

4. For decision trees, because it will keep dividing the training set until all the leaves are pure. That's why it got a very good performance on the training set. And it also overfitting, so the validation set performance is not good.

5. For Ada Boost, the result looks terrible. Since it only trains a tree stump (depth of 2) for each node, it can only divide the tree by one of the 60000 features. This model works well for small number of features, but it is not a fitful model for this dataset.

7. For Random Forest, each tree will be divided until pure. It is useful especially for the high dimensional data. That's why it looks much better than Tr Ada Boost.

6. It is not surprising that Naïve Bayes model got a good performance on both the training set and the validation set. Because this model only depends on the probability of the features according to it class.

7. Multiclass Logistic Regression model and Multilayer Perceptron use the same idea to some extent. They are the base of deep neuron network. In my result, the Multilayer Perceptron which has 256 nodes in the hidden layers is worse than 100 nodes, that is because I set the maximum iteration. For 256 nodes in the hidden layers, it is not convergence yet. But to make this comparison fair, I need to stop the training.

4. Final Results and Interpretation

I will only use two baseline models (trivial one and SVM with linear kernel) to make the comparison with the best model, which is Multilayer Perceptron with 100 nodes in the hidden layers. Use adam for gradient descent strategy, learning rate = 0.0001, batch size = 200, epoch = 100, use ReLU as activation function. This time, I train these 3 models on the whole training set, and test it on the test set.

model	training accuracy	test accuracy
trivial	0.053	0.053
non_trivial	0.999	0.629
MLP	0.999	0.771

From the result, we can find that the accuracy of both the MLP and not trivial models decrease from the validation set, I guess maybe the training set is slightly different from the test set.

The out-sample error bound is:

$$E_{out}(h) \leq E_D(h) + \sqrt{\frac{1}{2N} ln \frac{2CM}{\delta}}$$

In this case, C = 20, let's suppose δ = 0.1.

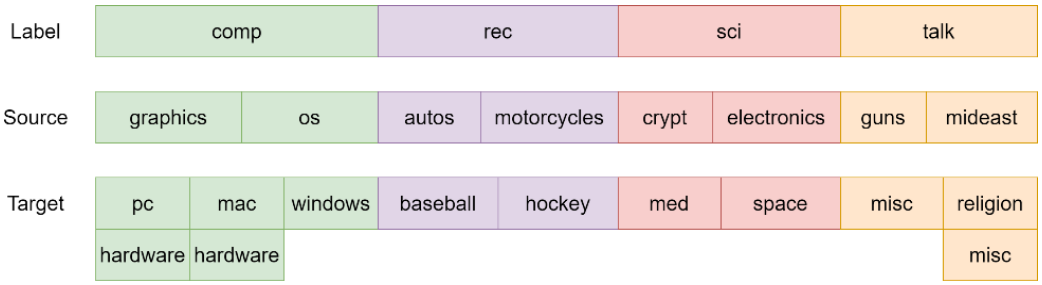
In the validation step, N = 9016, M = 24. So, the bound is 0.023

In the test step, N = 11269, M = 1. So, the bound is 0.016

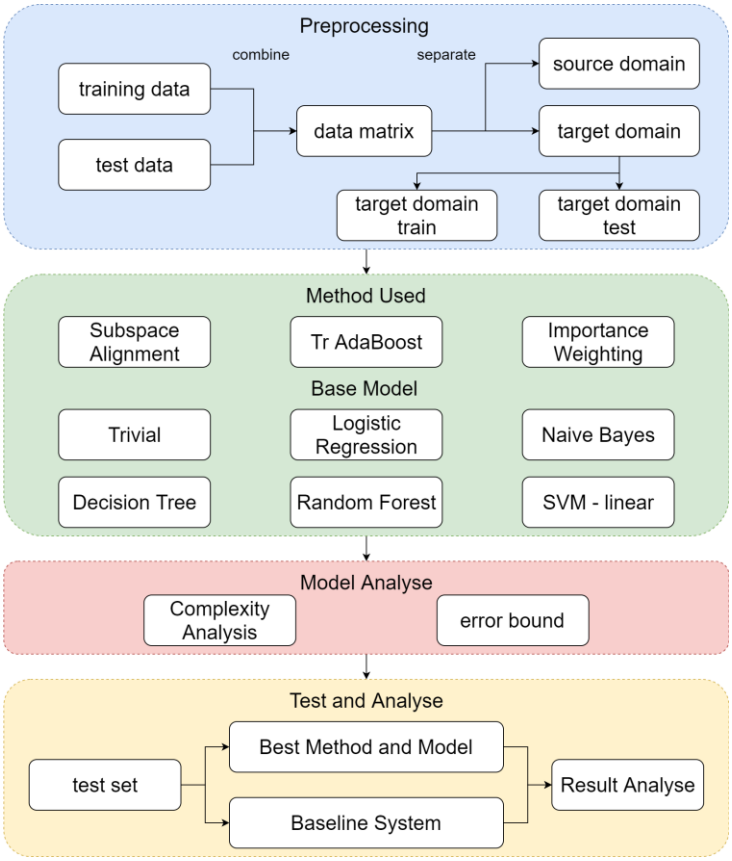
5. Implementation for the extension – Transfer Learning

5.1. Data Set

I will use "comp", "rec", "sci", "talk" as 4 top layer label, and label the corresponding data points as 0, 1, 2, and 3. For each top layer group, I will select the first two sub-group as source domain data, and the remaining as the target domain data.



5.2. Dataset Methodology



Before training, I will first combine the training set and the test set together, because in transfer learning problem. Then, separate 50 percent of target domain data use to train, and another 50 percent of target domain data use to test. To find the most fitful transfer learning strategy, and its corresponding model, I will divide a group of source domain data and target domain data.

5.3. Preprocessing, Feature Extraction, Dimensionality Adjustment

Since these is an inductive training model, the source domain and target domain data have its own unique features. For example, some words are only existed in the source domain and never exist in the target domain, and vice versa.

In statistics, there are total 61189 features for each data. 41799 features are used in the source domain (which means the feature's value is not equal to 0 for at least one data), and 43969 features are used in the target domain. 27364 features are used for all the domain, and 58404 features are used in at least one domain.

When we do transfer learning, the feature that only exist in one domain can't be used, we need to drop those features. And, to ensure that there is enough information to transfer, I only keep the feature that the sum value of it is greater than 50 for at least one domain, that means. For example, word "the" used in source domain 200 times, greater than 50, then I will keep it. Word "method" used in source domain 20 times, but used in target domain 51 times, greater than 50, then I will also keep it. Word "exception" used in source domain 40 times, and in target domain 45 times, I will drop it.

After this process, 5212 features are selected as my training features.

5.4. Training Process

All of the method below is implemented by adapt, which is a python library.

1. Baseline system: Train on the source domain and use it to predict the label on the target domain. When training on the source domain, the target domain data will not be used to help training.

   To calculate the time complexity, let's suppose there are N_s data points for source domain, N_t data points for target domain, D features, and C classes. For baseline system, all the model's complexity are the same as supervise learning. The time complexity for SVM, Logistic Regression, and Naïve Bayes is O(NsDC), for Decision Tree is $O(N_s\log(N_s)D)$, and for Random Forest is $O(MN_s^{0.5}\log(N_s)D)$, where M is the number of trees.

2. Subspace Alignment: This method is trying to find the distribution pattern of the source domain data and the target domain data. Then, rotate the source domain data to make it aligned with target domain data. Typically, Subspace Alignment is not a fitful method for this dataset.

   SA will first do PCA for both the source domain and the target domain and then transform the data, finally, do the same process as baseline system. The time complexity for PCA is $O(D^2N+D^3)$ [7]. In our case, we calculate PCA for both domains, the time complexity for PCA is $O(D^2(N_s+N_t)+D^3)$. The transformation is also $O((N_s+N_t)D^2)$. So, the total extra time complexity for SA is $O(D^2(N_s+N_t)+D^3)$.

3. Tr Ada Boost: The main purpose of Tr Ada Boost is to find the weight factor for all the training data point. For those data points that are less important in the target domain, reduce the weight factor. And for those data points that are important in the target domain, increase the weight factor. The default iteration is 10, but Tr Ada Boost is easy to overfitting, hence, I set the iteration equal to 5.

   For K iteration Tr Ada Boost, it will use K base models to classify, and for each model, it uses source domain and target domain together. Therefore, the time complexity for Tr Ada Boost is vary model by model. It also needs to iterate through all the data points to update the weight. For SVM, Logistic Regression, and Naïve Bayes is $O(K(N_s+N_t)DC)$, for Decision Tree is $O(K(N_s+N_t)\log((N_s+N_t))D)$, and for Random Forest is $O(KM(N_s+N_t)0.5\log((N_s+N_t))D)$, where M is the number of trees.

4. Importance Weighting: Importance Weighting is not used in the problem since it needs to know the probability of the data distribution. If we don't know the data distribution, we need to estimate them by the data we have. We also need to reduce the feature as I mentioned above to ensure we have enough data to estimate the parameter of gaussian distribution. Also, we need to assume that the features are independent to each other, which is like Naïve Bayes to some extent.

   But after doing some experiments, the result still can't be used. Because the input data is a sparse matrix, most of the features for one data is 0, only a few features have positive number. When estimate the parameter of gaussian distribution, it is easily to get something like $\theta = 0.1$ and $\sigma = 0.1$. When multiply those 5212 features together, and get the final weight ratio, is always equal to 0 or infinite. Which means all the features that only in the target domain is infinite important, and the features that only in the source domain is zero important.

   Obviously, this method is not fitful for this problem. I will not use this method.

5.5. Model Selection and Comparison of Results

SA method cannot use Naïve Bayes, because Naïve Bayes request all the input data should be positive, however, when doing the transformation, some of the data will become negative. Therefore, there are no Naïve Bayes result in SA.

method	model	comp	rec	sci	talk	sum
Baseline	MLR_OVR	0.627	0.400	0.441	0.695	0.540
	SVM_linear	0.612	0.435	0.410	0.621	0.522
	MLR_OVR	0.627	0.400	0.441	0.695	0.540
	Naive_Bayes	0.801	0.764	0.362	0.708	0.670
	decision_tree	0.451	0.204	0.397	0.541	0.395
	random_forest	0.666	0.329	0.431	0.715	0.537
SA	SVM_linear	0.461	0.422	0.446	0.705	0.489
	MLR_OVR	0.530	0.418	0.345	0.801	0.504
	decision_tree	0.311	0.289	0.286	0.325	0.302
	random_forest	0.439	0.270	0.222	0.410	0.342
Tr Ada Boost	SVM_linear	0.882	0.911	0.812	0.819	0.861
	MLR_OVR	0.915	0.931	0.861	0.866	0.898
	Naive_Bayes	0.970	0.977	0.832	0.913	0.929
	decision_tree	0.688	0.734	0.599	0.571	0.657
	random_forest	0.967	0.943	0.846	0.840	0.911

From the result, I conclude that:

1. The classification accuracy for different group is vary model by model. When use baseline transfer learning (without using transfer learning) with SVM linear kernel, the highest accuracy is "talk" group. However, in Naïve Bayes, the highest accuracy group is "computer". Therefore, we can't say which group's source domain and target domain is closer to each other.

2. SA method is always worse than baseline. Because SA is good at handling low dimension data, and it need some labeled target domain data to help it "flip" the prediction. In this problem, the data is high dimensional, it is not suitable for this method. Adjust the data distribution in the source domain will make the model predict the wrong result.

3. From the table above, Tr Ada Boost is always better than baseline. Because Tr Ada Boost is the only model here using target domain data and label to help it adjust the weight value. It is target oriented, always use target data and label to make the model perform better on the target domain.

6. Final Results and Interpretation for the extension

Only consider the error bound on the best performance model, which is using Tr Ada Boost and Naïve Bayes.

The out-sample error bound is:

$$e_T(h) - e_T(e_T^*) \leq 2(1-\alpha)\left[e_{S,T}^* + \frac{1}{2}d_{H\Delta H}(P_S, P_T)\right] +$$

$$4\sqrt{\frac{\alpha^2}{\beta} + \frac{(1-\alpha)^2}{1-\beta}}\sqrt{\frac{2}{N}d_{vc}\,ln[2(N+1)] + \frac{2}{N}ln\frac{8}{\delta}}$$

Where α: importance of error in target domain, suppose it is 0.5

β: fraction of labeled data points drawn from target domain, in this case is 7735/ (7735+4140) = 0.651

$e_{S,T}^*$: is the combined error = $\min_{h\in H}\{e_S(h) + e_T(h)\}$. I retrain and make prediction on the source and target domain again, and get 0.255.

$d_{H\Delta H}(P_S, P_T)$: Symmetric difference hypothesis divergence = 2 $\sup_{hh'\in H}\{|P_S[h\neq h'] - P_T[h\neq h']|\}$, suppose it is 0.1 in this case.

In this case, N = 16015, d_{vc} = 2 (total 4 classes), let's suppose δ = 0.1.

The final error bound is: 0.54.

7. Contributions of each team member

I did this project along, and I have no other team member.

8. Summary and conclusions

Text data is completely different with all other data I have handled before. It is sparse and high dimensional, which require some new strategy to deal with. Over all the models I use, whatever in the supervise learning part or the transfer learning part, SVM with linear kernel, Naïve Bayes, Random Forest, and Logistic Regression is always good choices for handling text data.

In this project, I only use one-hot encoding method to preprocess the text data, in the future, I may try other word embedding method in NLP and see whether it perform better or not.

9. References

[1] "MOST POPULAR WORD EMBEDDING TECHNIQUES IN NLP" 18 August 2020. [Online]. Available: https://dataaspirant.com/word-embedding-techniques-nlp/

[2] 20news dataset [Online]: http://qwone.com/~jason/20Newsgroups/

[3] Time complexity for SVM [Online]: https://www.quora.com/What-is-the-computational-complexity-of-an-SVM

[4] Time complexity for Logistic Regression [Online]: https://levelup.gitconnected.com/train-test-complexity-and-space-complexity-of-logistic-regression-2cb3de762054

[5] Time complexity for Logistic Regression [Online]: https://levelup.gitconnected.com/train-test-complexity-and-space-complexity-of-logistic-regression-2cb3de762054

[6] Time complexity for Decision Tree [Online]: https://towardsdatascience.com/almost-everything-you-need-to-know-about-decision-trees-with-code-dc026172a284

[7] Time complexity for PCA [Online]: https://stackoverflow.com/questions/20507646/how-is-the-complexity-of-pca-ominp3-n3

10. Appendix

Appendix I.

extracted zip folder:

```
├── main.py
├── README.md
├── load_data.py
├── statistic.py
├── SL.ipynb
├── TL.ipynb
├── 20news/matlab
│   ├── test.data
│   ├── test.label
│   ├── test.map
│   ├── train.data
│   ├── train.label
│   ├── train.map
│   └── vocabulary.txt
├── model
│   ├── SL
│   │   ├── final_MLP.joblib
│   │   └── final_SVM_trivial.joblib
│   ├── TL
│   │   ├── Final_Tr_Ada_Boost.joblib
│   │   └── SA_SVM_linear.joblib
```