Project Machine Learning Reports for Milestone 2

Zhanwang Chen

Classification methodology

Explain what you have done.

1. A short description of the classification methods (not more than two pages).

SVM

The goal of SVM is to find a hyperplane that would leave the widest possible "cushion" between input points from two classes.

Random forests

Random forests are a combination of tree predictors such that each tree depends on the values of a
random vector sampled independently and with the same distribution for all trees in the forest.

Random Forest creation pseudocode:
1.Randomly select “K” features from total “m” features where k << m
2.Among the “K” features, calculate the node “d” using the best split point
3.Split the node into daughter nodes using the best split
Repeat the a to c steps until “l” number of nodes has been reached
Build forest by repeating steps a to d for “n” number times to create “n” number of trees

random forest prediction pseudocode is shown below:
Takes the test features and use the rules of each randomly created decision tree to predict the outcome and stores the predicted outcome (target)
Calculate the votes for each predicted target
Consider the high voted predicted target as the final prediction from the random forest algorithm

CNN

Pad each sentence to the maximum sentence length, here we set it to be 50. Then append special <PAD> tokens to all other sentences to make them 50 words. This allows us to efficiently batch our data, since each example in a batch must be of the same length.
Build a vocabulary index and map each word to an integer between 0 and the vocabulary size. Each sentence becomes a vector of integers.

Dropout Layer
it stochastically “disables” a fraction of its neurons. This prevent neurons from co-adapting and forces them to learn individually useful features. The fraction of neurons we keep enabled is defined by the dropout_keep_prob input to our network. We set this to something like 0.5 during training
pretrained German Wikipedia word embedding was used for initialize the embedding layer.

The CNN model here was modified from the Implementation [8] as paper [7],

Naive Bayes classifier

The Naive Bayes classifier assumes that the features used in the classification are independent. since it can be trained very quickly and simple, so it is used here as a baseline.

It uses the words (or terms/tokens) of the document in order to classify it on the appropriate class. By using the “maximum a posteriori (MAP)” decision rule

$$c_{map} = \arg\max_{c \in C}\left(P(c \mid d)\right) = \arg\max_{c \in C}\left(P(c)\prod_{1 \le k \le n_d} P(t_k \mid c)\right)$$

Where t_k are the tokens (terms/words) of the document, C is the set of classes that is used in the classification, the conditional probability of class c given document d, the prior probability of class c and the conditional probability of token t_k given class c.

2. **A description of the chosen pre-processing and feature selection (you can refer to your report from Milestone 1). Take optimal choices for each of the assigned classification methods separately.**
   Bag of word, tfidf, different percents of feature selection, and the different feature selection method(MI,chi2 and so on) as reported in Milestion1 was tried for naïve Bayesian, svm, random forest.
   word embedding ,tfidf, bag of word features for CNN model was evaluated.
   For svm, word emending features performs best.
   For naïve Bayesian bag of word performs best.
   For random forest bag of word performs best
   For CNN, word emending features performs best.
   Since different classes’ data are rather uneven, StratifiedShuffleSplit was used in Gridseach because its folds are made by preserving the percentage of samples for each class.
3. **Explain the meaning of the hyperparameters and how you have chosen them (model selection).**

SVM

C controls the cost of misclassification on the training data.

The goal of SVM is to find a hyperplane that would leave the widest possible "cushion" between input points from two classes. There is a tradeoff between "narrow cushion, little / no mistakes" and "wide cushion, quite a few mistakes".

$$\begin{aligned}
\min_{\mathbf{w},\boldsymbol{\xi}} \quad & \tfrac{1}{2}\mathbf{w}^t\mathbf{w} + C\mathbf{1}^t\boldsymbol{\xi} && (1)\\
\text{subject to} \quad & y_i(\mathbf{w}^t\phi(\mathbf{x}_i) + b) \geq 1 - \xi_i && (2)\\
& \boldsymbol{\xi} \geq 0 && (3)\\
\text{where} \quad & \phi^t(\mathbf{x}_i)\phi(\mathbf{x}_j) = K(\mathbf{x}_i, \mathbf{x}_j) && (4)
\end{aligned}$$

if the error penalty C is high, it will end up with less misclassifications and fewer support vectors (the highlighted points are SVs). if too high when using non-linear kernels it might overfit.

kernel

The RBF kernel on two samples **x** and **x'**, represented as feature vectors in some *input space*, is defined as[2]

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

An equivalent, but simpler, definition involves a parameter :

$$K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)$$

the gamma parameter is the inverse of the standard deviation of the RBF kernel (Gaussian function), which is used as similarity measure between two points. Intuitively, a small gamma value defines a Gaussian function with a large variance.

polynomial kernel represents the similarity of vectors (training samples) in a feature space over polynomials of the original variables, allowing learning of non-linear models.

The polynomial kernel looks not only at the given features of input samples to determine their similarity, but also combinations of these.

For degree-d polynomials, the polynomial kernel is defined as

$$K(x, y) = (x^{\mathsf{T}} y + c)^d$$

where x and y are vectors in the input space. c ⩾ 0 is a free parameter trading off the influence of higher-order versus lower-order terms in the polynomial. d is the degree of the polynomial

mapping. The most common degree is d = 2 (quadratic), since larger degrees tend to overfit on NLP problems.

The linear kernel is a special case of polynomial_kernel with degree=1 and coef0=0.

Random Forest

max_features:

These are the maximum number of features Random Forest is allowed to try in individual tree. Increasing max_features makes the model has a higher number of options to be considered. It also decrease the speed of algorithm by increasing the max_features.

n_estimators :

This is the number of trees you want to build before taking the maximum voting or averages of predictions. Higher number of trees give you better performance but makes the model slower.

min_sample_leaf :

Leaf is the end node of a decision tree. A smaller leaf makes the model more prone to

capturing noise in train data.

CNN

filter_sizes – The number of words we want our convolutional filters to cover.

num_filters – The number of filters per filter size

different word embedding model, for example, word2vec, fastText from Facebook.

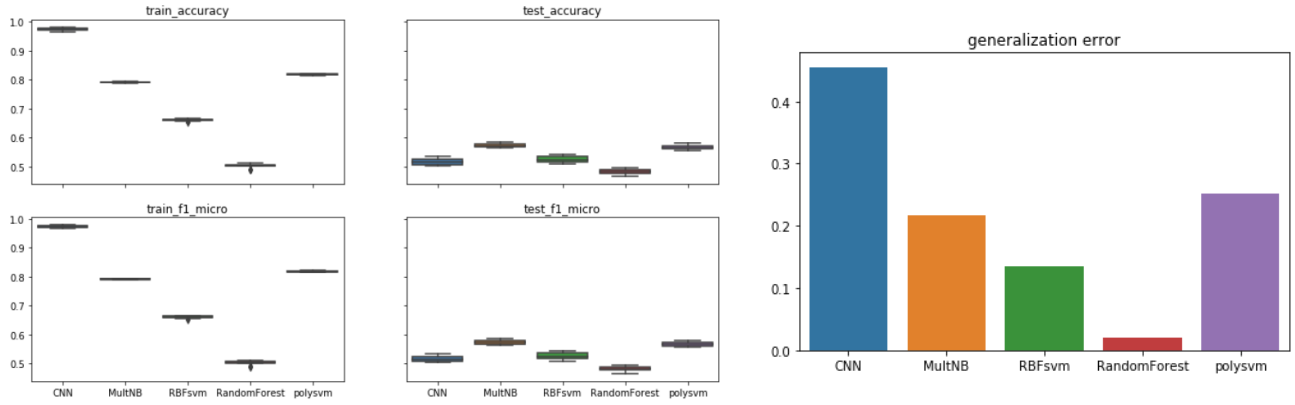
Empirical estimate of the generalization error

1. Compute estimates of the generalization error (using some kind of error bars), i.e. the expected error rates on new data (not used during training).

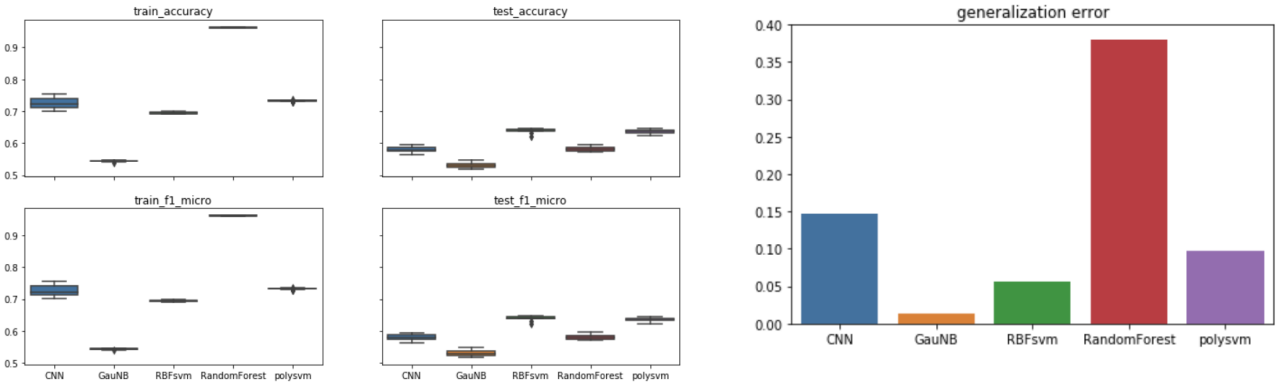
CNN generalizes not that well, validation accuracy usually 20% lower than that in training.

In generalization error plot, y axis is (training accuracy – test accuracy), eg. Bag of word: RBFSVM generalization error = training accuracy 0.65– test accuracy0.5=0.15

For bag of word features:



For word Embedding features:



Since MultiNB can not be used for negative values, Gaussian NB was used.

4. What is a good error measure for this multiclass problem?

As in Milestone1 we can the data size of each class is rather uneven, F1_micro can be a good error measure, and it also make it easy to plot the ROC curve.

In “micro averaging,” it computes the true positives, false positives, of the k-class model, then

$$PRE_{micro} = \frac{TP_1 + \cdots + TP_k}{TP_1 + \cdots + TP_k + FP_1 + \cdots + FP_k}$$

And in macro-averaging, we average the performances of each individual class:

$$PRE_{macro} = \frac{PRE_1 + \cdots + PRE_k}{k}$$

3. For this application, both the true positive rate and the false positive rates are important. Which trade-offs are achievable?

A ROC(receiver operating characteristic) space is defined by FPR(false positive rates) and TPR(true positive rate) as x and y axes, respectively, which depicts relative trade-offs between true positive (benefits) and false positive (costs). any increase in sensitivity will be accompanied by a decrease in specificity. Since TPR is equivalent to sensitivity and FPR is equal to (1 – specificity). If we say both the true positive rate and the false positive rates are important. We can chose the intercept of the ROC curve with the line at 45 degrees orthogonal to the no-discrimination line - the balance point where Sensitivity = Specificity,as Figure 2 Shows, point A.

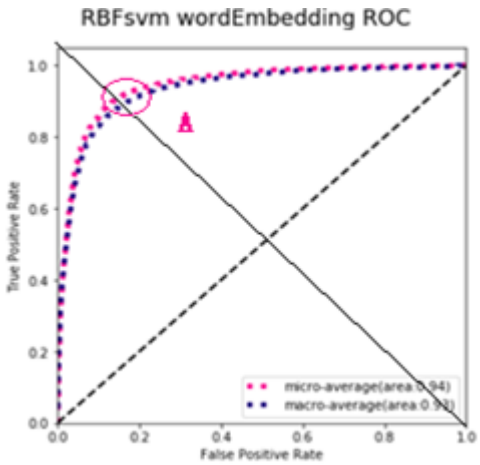


Figure 2

Discussion

1. **How expensive is it to train the classifiers? Is it possible to efficiently re-train the classifiers as new data becomes available?**

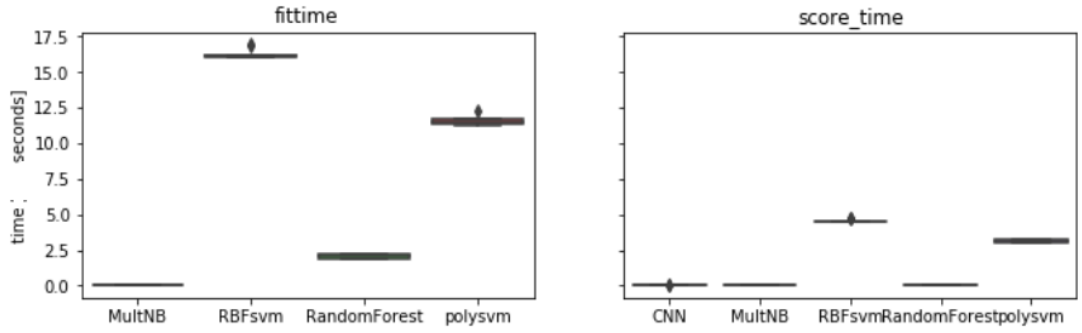
   StratifiedShuffleSplit() with 10 splits was used to test the training and testing.

   For CNN, is not show in the plot, because it took much longer than the rest of the models. It took 7 seconds per epoch, and 116 minutes to finish 1000 epochs that reach 81% accuracy. Score time is time that finish all the test set which is 20% of the whole dataset.

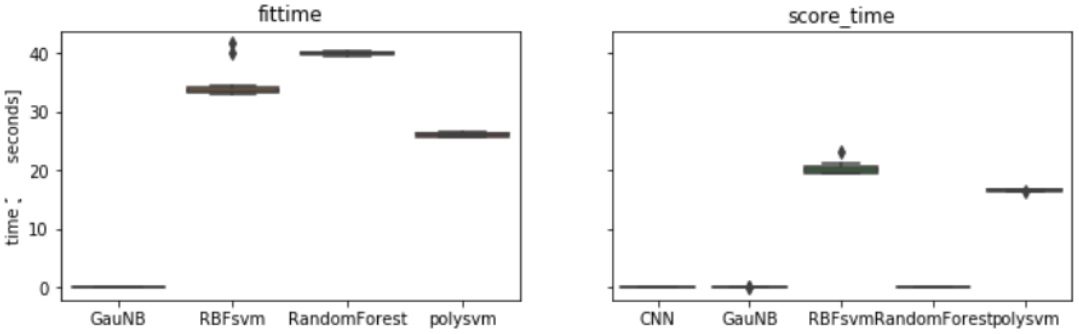
   Naïve Bayesian models perform super fast for both training and predicting. RBF SVM is quite slow for both training and predicting. Here word embedding has 300 fixed

dimensions

For bag of word features:



for word embedding features:



Only MultinomialNB in scikit-learn support Incremental learning ,since it implemented the partial_fit() API

For svm, there is an extension[9] of LIBLINEAR supports incremental and decremental learning

For ramdomForest, Online Random Forests[10] can be used.

For CNN, method model.save() can be used to save the model's weights, architecture and so on, after new data come, we can use load_model() to continue training.

2. **Is it possible to obtain a confidence measure for the predictions using these prediction algorithms?**

   For SVM, in scikit-learn, set parameter "probability" to True to enable probability estimates. it must be enabled prior to calling fit, but will slow down that method. Then predict_proba() method can be used to
   compute probabilities of possible outcomes for input samples.

   For ramdomForest, method predict_proba(X) can be used to predict class probabilities for input X.
   The predicted class probabilities of an input sample are computed as the mean predicted class probabilities of the trees in the forest. The class probability of a single tree is the fraction of samples of the same class in a leaf.

Multinomial Naive Bayes and Gaussian Naive Bayes can use predict_proba(X) to compute probability estimates for the test vector X.

For CNN model, in Keras, method predict_proba() can be used to generate class probability predictions for the input samples.

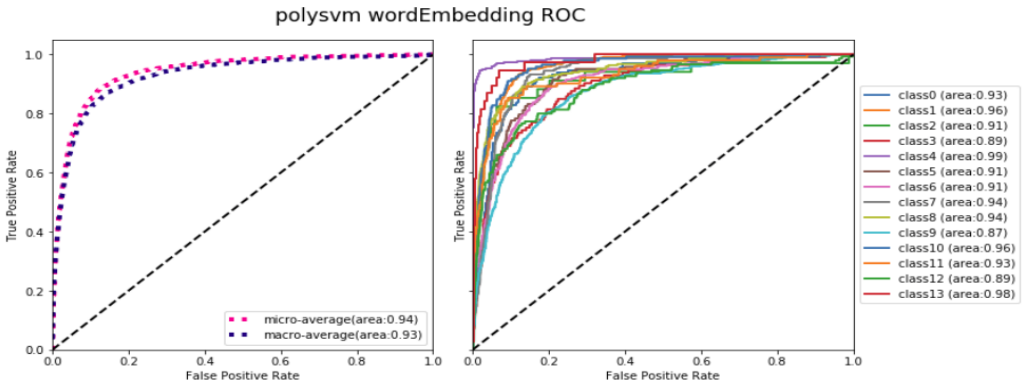
3. **Draw a conclusion: are the learning algorithms suitable for this application? Which one is better?**

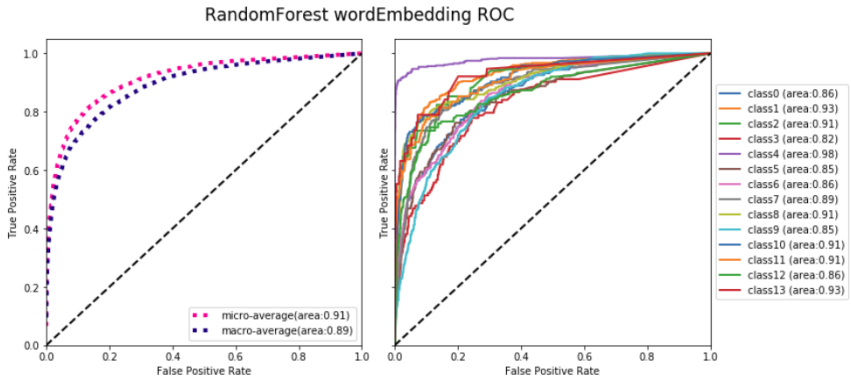
Different ROCs can be seemed in appendix.

After compare with different model with different features, SVM with RBF kernel, fastText word embedding performs best, in terms of F1_micro score and generalize error. Poly and RBF kernel SVM have the highest F1_micro score.

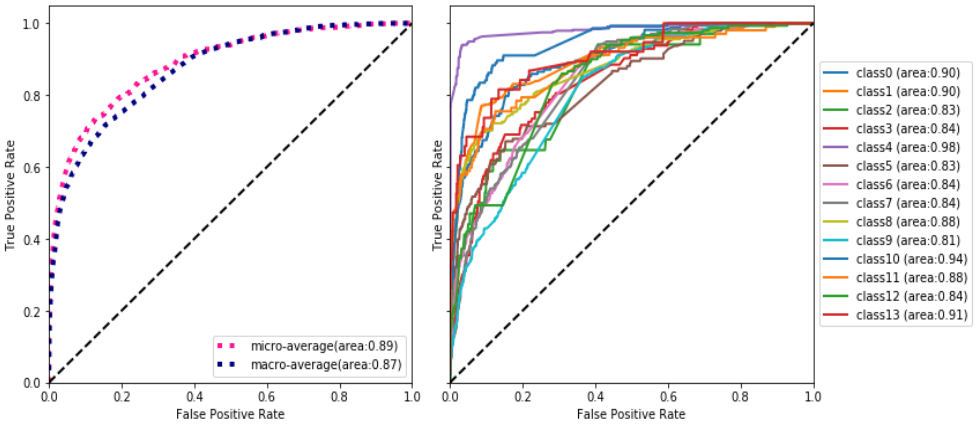
Appendix

Some of the ROCs:

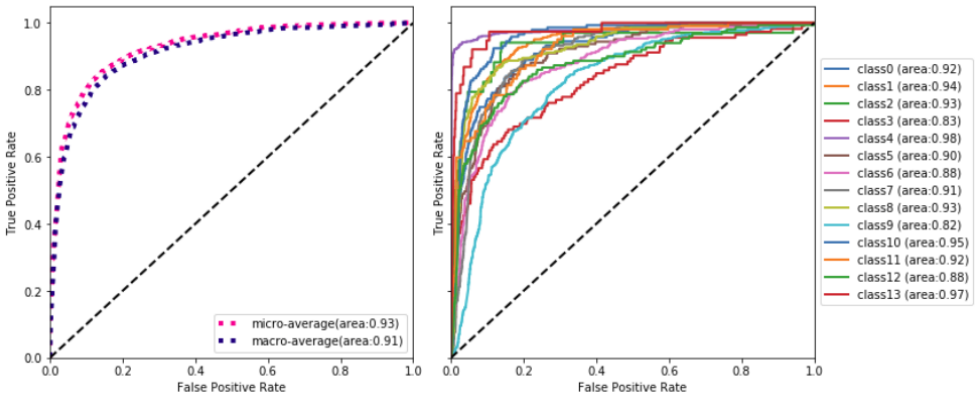




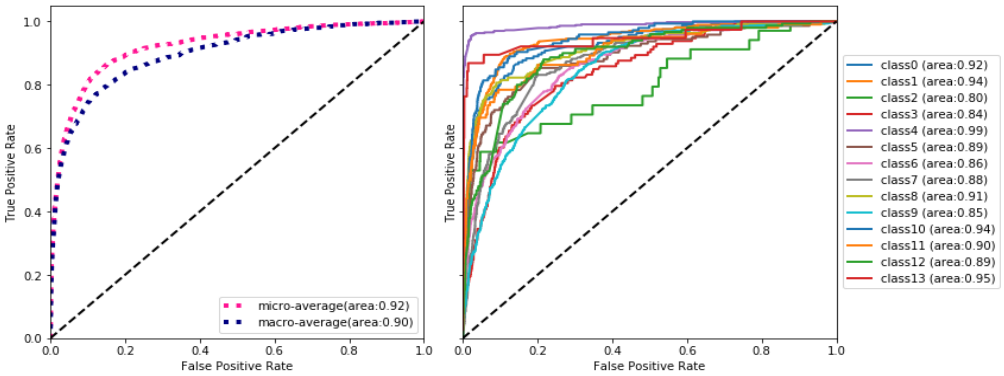
RandomForest bag of word ROC

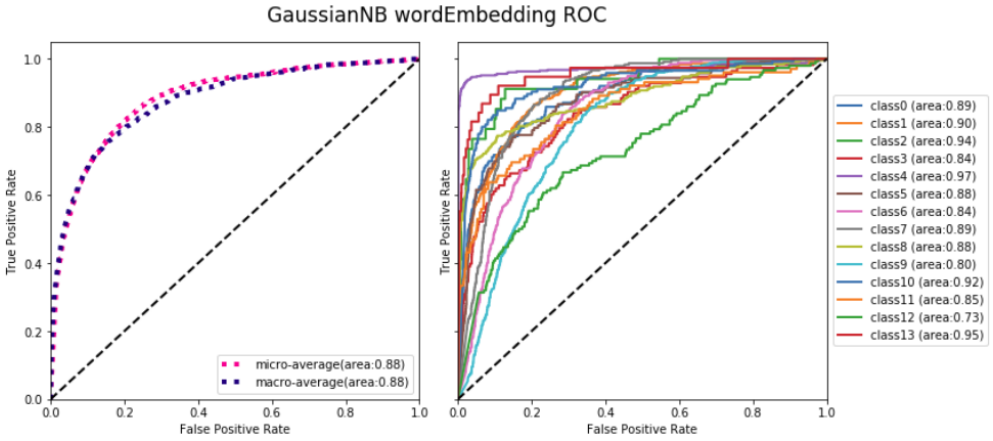


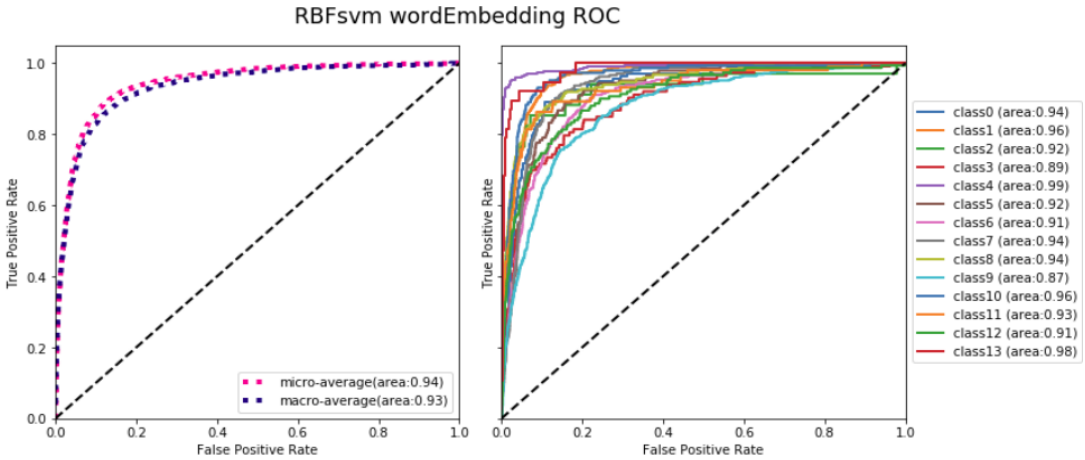
CNN wordEmbedding ROC



polysvm bag of word ROC







REFERENCES

[1] **https://en.wikipedia.org/wiki/Radial_basis_function_kernel**

[2] **http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html**

[3] **https://en.wikipedia.org/wiki/Polynomial_kernel**

[4] Training and Testing Low-degree Polynomial Data Mappings via Linear SVM

[5] Random Forests LEO BREIMAN

[6] https://link.springer.com/content/pdf/10.1023%2FA%3A1010933404324.pdf

[7] Convolutional Neural Networks for Sentence Classification Yoon Kim

[8] https://github.com/alexander-rakhlin/CNN-for-Sentence-Classification-in-Keras

[9] https://www.csie.ntu.edu.tw/~cjlin/papers/ws/index.html Incremental and Decremental Learning Extension of LIBLINEAR

[10] https://github.com/amirsaffari/online-random-forests Online Random Forests