CS294-1 Final Project

Algorithms Comparison

Deep Learning Neural Network — AdaBoost — Random Forest

Prepared By:
Shuang Bi (24094630)
Wenchang Zhang (24094623)

2013-05-15

1 INTRODUCTION

In this project, we are going to apply three different algorithms on different datasets. They are deep learning neural network, adaptive boosting, and random forest. Our goal is to find the rule of thumb of selecting suitable algorithms for different datasets and to analyze different characteristics for the three algorithms. We make use of various programming tools including packages in Java and R. We will measure the performance of different models by performing 10-fold cross validation.(We did not do cross validation on Reuters data)

2 DATA

The datasets we will be using in this project are SMS spam message collection, Farm Advertisement and Reuters Text Categorization Collection Data taken from UCI machine learning repository. Now we will discuss their formats and information we extracted separately in the subsections below.

2.1 SMS Spam Collection

The SMS spam collection data set contains a set of labeled messages that are collected from a UK forum. The data set consists of 13.4% spam messages and 86.6% ham messages (useful messages). Examples of spam and ham message are shown in Table 1 below.

Table 1: Example of Spam and Ham Message

	Example
spam	FreeMsg: Txt: CALL to No: 86888 & claim your reward...
ham	What you doing? how are you?

After observing patterns of a sample of spam messages and doing some research on key features of identifying spam messages, we decide to use the features that are specified below in our model.

- Currency sign count; website address count; hash ID count; words with length less than 3 counts; white space count
- Frequency of letters (upper case and lower case); frequency of numbers; frequency of punctuations and other special characters;

- Ration of capital words (words contain capital letters)
- Average word length; average sentence length in characters; average sentence length in words
- One-gram high frequency keywords count; two-gram high frequency keywords count; tri-gram high frequency keywords count

Note that the one/two/tri-gram words are extracted from the text. For example, the two-gram words are extracted by looking at all the possible two-word combinations within a group of 10 words taken in the order of the sentence. After obtaining all the possible one/two/tri-gram words, we get the top-occurrences spam words. We named them as our one-gram, two-gram, tri-gram keywords to identify the spam messages. We used the java package *corpuslinguistics* to do the ngram extraction. Finally we extract 64 original features for each short message.

2.2 Farm Advertisement

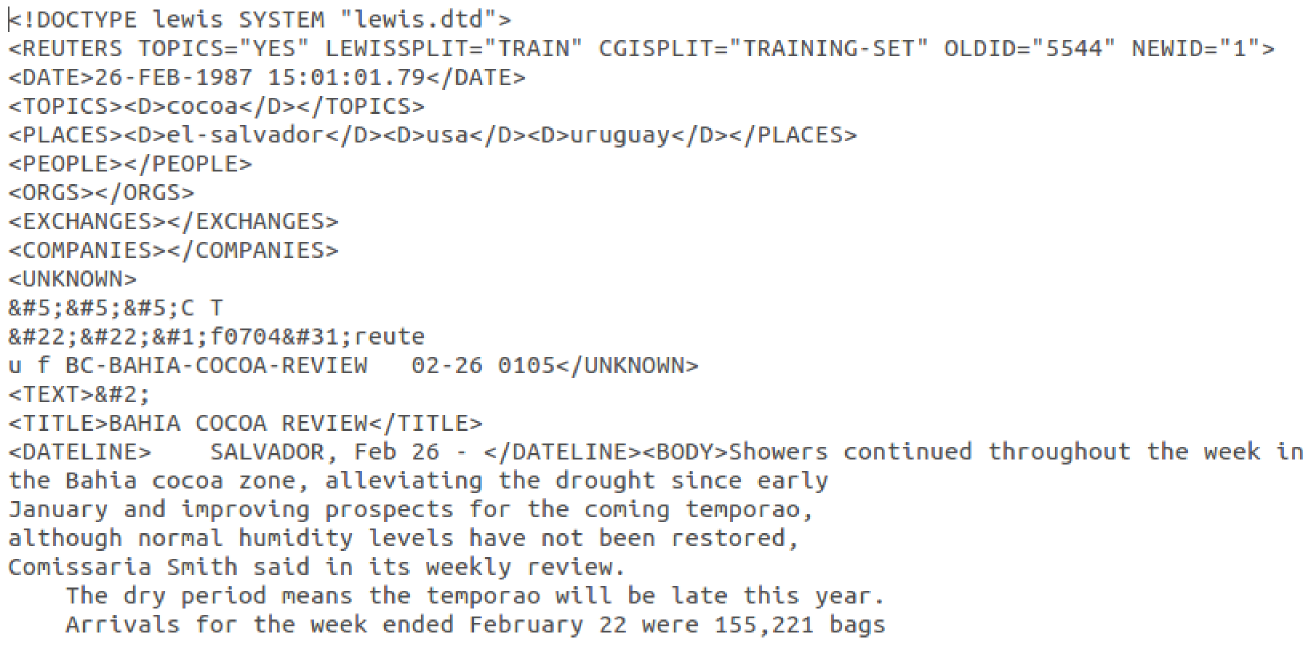
This data set was collected from text ads found on 12 websites that are based on various farm animals. The text ads are labeled with binary tags with 1 representing accepted ads and -1 representing rejected ads. There are 53.3% accepted ads and 46.7% rejected ads. The data set is already stemmed and the stop words are removed. The data set only contains words and they are not in any order, so the only features we can get from this data are the words themselves. Finally we extract 66 original features for each advertisement.

2.3 Reuters Reuters Text Categorization Collection

The documents in the Reuters collection appeared on the Reuters newswire in 1987. The original data are saved in SGML files, and the snapshot of the data are as follows Since there are many effective tools to parse XML file instead of SGML, we wrote Java code to convert SGML to XML first, then applied Java *javax.xml.parsers* package to parse XML. In the Reuters data, each article may fall into more than 1 topics, or no topics at all. Therefore we only considered top 6 most frequent topics, which are

- acq, trade, grain, crude, earn, money-fx

Thus we have 6 0-1 values as responses for each article. Then we treat high frequency words in title and body of text seperately as features. To do that, we built



```
<!DOCTYPE lewis SYSTEM "lewis.dtd">
<REUTERS TOPICS="YES" LEWISSPLIT="TRAIN" CGISPLIT="TRAINING-SET" OLDID="5544" NEWID="1">
<DATE>26-FEB-1987 15:01:01.79</DATE>
<TOPICS><D>cocoa</D></TOPICS>
<PLACES><D>el-salvador</D><D>usa</D><D>uruguay</D></PLACES>
<PEOPLE></PEOPLE>
<ORGS></ORGS>
<EXCHANGES></EXCHANGES>
<COMPANIES></COMPANIES>
<UNKNOWN>
&#5;&#5;&#5;C T
&#22;&#22;&#1;f0704&#31;reute
u f BC-BAHIA-COCOA-REVIEW   02-26 0105</UNKNOWN>
<TEXT>&#2;
<TITLE>BAHIA COCOA REVIEW</TITLE>
<DATELINE>    SALVADOR, Feb 26 - </DATELINE><BODY>Showers continued throughout the week in
the Bahia cocoa zone, alleviating the drought since early
January and improving prospects for the coming temporao,
although normal humidity levels have not been restored,
Comissaria Smith said in its weekly review.
    The dry period means the temporao will be late this year.
    Arrivals for the week ended February 22 were 155,221 bags
```

Figure 1: Snapshot of Reuters Text Categorization Collection

a dictionary for words in titles and a dictionary for words in bodies of texts. Our design matrix for this dataset contains 847 columns, 200 of which are for words in titles and the rest are for words as well as frequency of appearance of numbers in bodies.

3 ALGORITHMS and REALIZATION

In this project, we aim to apply three different algorithms to the data sets specified above. We are going to explain how they work in detail in this section.

3.1 Deep Learning Neural Network

- K-classification, $Y_k, k = 1, 2, ..., K$, are $0-1$ variables
- Feature Z_m (hidden units) are created from

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X), m = 1, 2, ..., M$$

  where $\sigma(v) = \frac{1}{1+e^{-v}}$

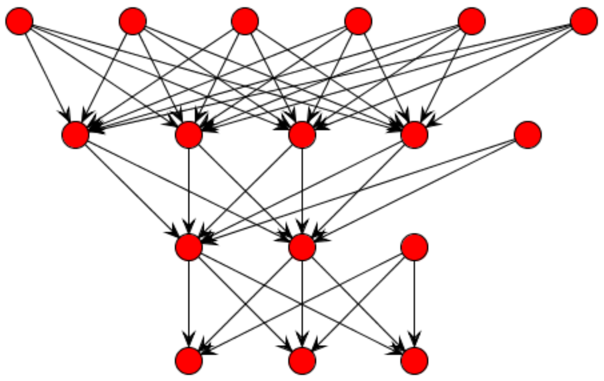


Figure 2: A Neural Network Structure with 2 Hidden Layers

- Fit hidden units to model

$$f_k(X) = g_k(\beta_{0k} + \beta_k^T Z)$$

where $g_k(T) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}}, k = 1, 2, ..., K$

Deep learning means bringing multiple layers instead of one to the structure. For all the three datasets, we used package *neuralnet* in R to do the fitting, and the number of layers in our study is up to 3. The two hidden layer neural network structure is shown in Figure 2.

3.2 Adaptive Boosting

AdaBoost adaptively weigh the data by assigning higher weight to the data points that are misclassified by the previous weak learner. In each iteration, we are going to learn a new weak classifier on the weighted dataset. After all the iterations, the weak classifiers are combined into a single strong classifier by assigning a different weight for each weak classifier based on their performance.

Let X represent the text body we are given, and Y represent the binary response. Therefore, $(x_1, y_1), ..., (x_m, y_m)$ are the data provided where $x_i \in X, y_i \in Y = \{-1, +1\}$

First initialized the distribution of the weights of all the data points: $D_1(i) = \frac{1}{m}$, where i = 1, ..., m. Next, we want to find the weak classifier for each iteration t, where t = 1, ..., T. Note that the classifier has to have an error rate that is less than 0.5. The classifier will read in the features (X), and output the binary response value. In another word,

$$h_t = \arg\min_{h_j \in H} \epsilon_j = \sum_{i=1}^{m} D_t(i)[y(i) \neq h_j(x_i)]$$

After we decide on which weak classifier we are going to use for a particular iteration, we will calculate the corresponding weight assigned to this classifier, named it α_t. Therefore, $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$.

We then update the weight for all the data points based on the performance of the classifier in the current iteration. The formula that is used for the update is,

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor which will make D_{t+1} indeed a distribution.

After we iterate t from 1 to T, we then output the weighted classifier and it will be our final strong classifier, H(x).

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$$

We will follow this rationale and apply it to our three data sets to build our model and make predictions. In practice we uesd the package *ada* in R to fit all the three dataset.

3.3 Random Forest

Random forest performs classifications by constructing a bag of unpruned decision trees. There are a lot of trees in this bag and each tree is grown based on a bootstrap sample from the given data set. The tree is grown by selecting a portion of the features at each node of the tree and choose the best split. The number of features selected for each node should be much smaller than the total number of features. Note that we do not perform any pruning during this process. The steps are as following,

1. Choose the number of input variables to be used to determine the decision at a node of the tree (m)

2. Generate bootstrap samples from the original data (i.e. sample with replacement from the data)

3. Train a decision tree for each given sample. There are M input variables/features, then randomly choose m << M variables at each node and choose the best split on these m variables

4. For prediction, the given sample will be pushed down the grown tree constructed based on the training data. The response variable will be the label of the terminal node it ends up with. We then iterate this procedure by all the trees grown and the response with more votes will be our final prediction

There is a package called *randomForest* in R to apply this algorithm. For the SMS collection and Farm advertisement, we used the *randomForest()* directly. However, for the Reuters data, due to the large number of features, we applied *parallel* package in R to define a parallel random forest function which fitted the data much faster than the original *randomForest()*.

4 RESULTS

In this section, we include RMSE results, accuracy rate for different algorithms and also accuracy v.s. folds plots in 10-fold cross validation. They will be shown based on different algorithms on the three data sets.

We use 10-fold cross validation to evaluate the performance of our model. We divide the dataset into 10 equal folds. We then use 9 folds as the training set and the remaining 1 fold as the testing set. Note that we acquired all of our results by performing 10-fold cross validation.

4.1 Deep Learning Neural Network

In this algorithm, we applied different number of units within a hidden layer to the SMS spam collection data set and different number of hidden layers to the Farm Ads data set.

First of all, we will list the RMSE values for the 10 different validations. The results are shown in Table 2 and 3

Table 2: RMSE for Deep Learning Neural Network (Spam Collection)

Fold as Testing	**1**	**2**	**3**	**4**	**5**
RMSE	0.1019	0.0898	0.0931	0.0913	0.0941
Fold as Testing	**6**	**7**	**8**	**9**	**10**
RMSE	0.0843	0.0914	0.0936	0.0918	0.0911

Table 3: RMSE for Deep Learning Neural Network (Farm Ads)

Fold as Testing	**1**	**2**	**3**	**4**	**5**
RMSE	0.3611	0.3584	0.3727	0.3942	0.3631
Fold as Testing	**6**	**7**	**8**	**9**	**10**
RMSE	0.3992	0.3804	0.3792	0.3580	0.3719

The average accuracy rate for different number of units for the first data set and for different number of hidden layers for the second are summarized in the Table 4 and 5 below. The tables also include the converging time in terms of number of iterations and running time for the two data sets.

Table 4: Accuracy Rate and Number of Iterations to Converge (Spam Collection)

# of Units	**Accuracy Rate**	**# of iterations**
u10	98.46%	646.5
u15	98.58%	985
u20	98.44%	894.5

Table 5: Accuracy Rate and Running Time (Farm Ads)

# of Hidden Layers	**Accuracy Rate**	**Time (s)**
1	92.78%	1044
2	93.06%	564
3	94.34%	253

The table above says that 15 units within a hidden layer works the best for the first data set and 3 hidden layers achieves a higher accuracy rate for the second data set.

Figure 3 illustrates the weighted features in our model for the spam message collection data set. The biggest ball corresponds to the average sentence length in words. Besides, number of words and number of high frequent unigram contribute important information, too.

Figure 3: Weighted Feature Visualization for Spam Collection

4.2 Adaptive Boosting

We repeat the same thing we did for the first algorithm. Table 6 and 7 are the RMSE calculations for the AdaBoost algorithm on the datasets.

Table 6: RMSE for AdaBoost (Spam Collection)

Fold as Testing	**1**	**2**	**3**	**4**	**5**
RMSE	0.1271	0.0947	0.1198	0.0947	0.1198
Fold as Testing	**6**	**7**	**8**	**9**	**10**
RMSE	0.1121	0.1641	0.0734	0.1198	0.1121

Table 7: RMSE for AdaBoost (Farm Ads)

Fold as Testing	**1**	**2**	**3**	**4**	**5**
RMSE	0.4313	0.3775	0.4313	0.3743	0.4053
Fold as Testing	**6**	**7**	**8**	**9**	**10**
RMSE	0.4199	0.3775	0.3870	0.4023	0.3962

The average accuracy rate for this algorithm will be summarized in the end, where the results for three algorithms are combined together for comparison.

4.3 Random Forest

Similarly, we repeat the statistics for random forest algorithm. They are shown in Table 8 and 9.

Table 8: RMSE for Random Forest (Spam Collection)

Fold as Testing	1	2	3	4	5
RMSE	0.1340	0.1271	0.1121	0.1271	0.1121
Fold as Testing	**6**	**7**	**8**	**9**	**10**
RMSE	0.1038	0.1121	0.1121	0.1038	0.1121

Table 9: RMSE for Random Forest (Farm Ads)

Fold as Testing	1	2	3	4	5
RMSE	0.4023	0.3743	0.3743	0.3870	0.3678
Fold as Testing	**6**	**7**	**8**	**9**	**10**
RMSE	0.3870	0.3775	0.3775	0.3901	0.3405

4.4 Overall Result

The average accuracy rate for three algorithms on three different data sets are shown below (Table 10 and 11). The accuracy v.s. fold number plots comparing different algorithms are also shown for the two data sets below (Figure 4, 5 and 6) . We also record the running time for the three algorithms on two data sets (Table 12).

Table 10: Accuracy Rate for Three Algorithms on SMS and Farm Ads Data

Algorithm	Spam Collection	Farm Ads
Neural Network	98.58%	94.34%
AdaBoosting	98.98%	84.03%
Random Forest	98.56%	85.65%

Table 11: Accuracy Rate for Three Algorithms on Reuters in terms of 6 Topics

Algorithm	acq	trade	grain	crude	earn	money-fx
Neural Network	99.88%	99.91%	99.93%	99.92%	99.74%	99.88%
AdaBoosting	97.93%	99.62%	99.76%	99.60%	99.13%	99.56%
Random Forest	97,68%	98.55%	99.44%	98.88%	97.69%	98.86%

Table 12: Running Time for Three Algorithms on Three Data Sets

Algorithm	Spam Collection (s)	Farm Ads (s)	Reuters (s)
Neural Network	159	476	1623
AdaBoosting	186	720	1482
Random Forest	148	342	972

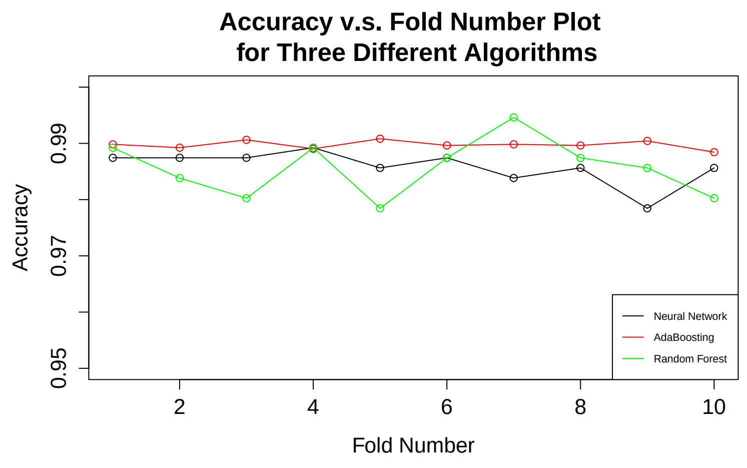


Figure 4: Accuracy Rate v.s. Fold Number for Spam Collection

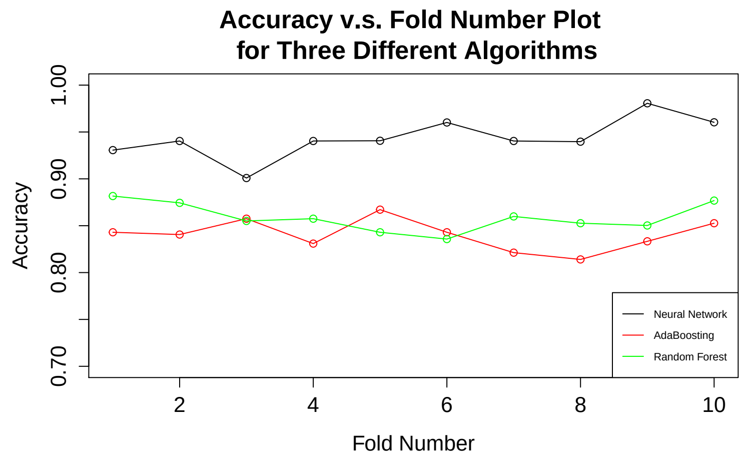


Figure 5: Accuracy Rate v.s. Fold Number for Farm Ads

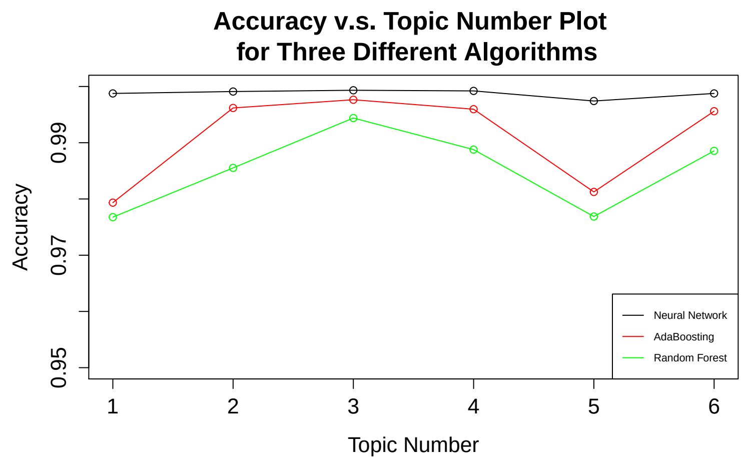


Figure 6: Accuracy Rate v.s. Topic Number for Reuters

5 CONCLUSION

From all the statistics provided above, we observe that for the spam collection data set, all algorithms perform well. They all get an average accuracy rate around 98%. The reason is that the data set contains all the useful information. It retains punctuations, special characters, capitalization of letters, etc. This characteristic allows us to extract dynamic features from the data set and make more accurate predictions.

However, for the second data set, deep learning neural network has the highest accuracy rate. The reason for this phenomenon is that the given data set in this case is already stemmed. All the information except the words are removed from this data set. The order of the words is destroyed as well. Therefore, the only features we can extract from the data are the words themselves. Applying AdaBoost or random forest algorithm can only build models based on the given words. On the other hand, deep learning neural network takes the inner links among features into consideration. By applying multiple hidden layers on our model, we would be able to capture the confounding relationships among the features and combine the features into strong features each time we apply the additional layer.

For the third data set, deep learning neural network outperform the other two in respect of accuracy. It is mainly because deep learning neural network seizes the information contained in the combination of words both in titles and bodies of texts. In respect of efficiency, these three algorithms takes less than 20 minutes for each topic. The random forest is the fastest because of the parallel computing.

Though random forest is faster than the other two, we observe that random forest has the biggest variance among the three algorithms.