CS155 Kaggle Competition Report

Team LNZ: Hongjian Lan*, Xiang Ni*, Taokun Zheng*

Abstract

(OVERVIEW) The major goal of this project is to predict financial recession given the frequencies of the top 500 word stems in the reports of financial companies. After applying various learning models, we can see that the prediction of financial recession by the bag of words has an accuracy of more than 90%. Hence, there is indeed a correlation between the two.

Moreover, we have compared different learning models (ensemble methods with Decision Tree, SVM, and KNN) with various parameters to find the best model with a relatively high average accuracy and low variance of accuracy by cross-validation on the training data set. In addition, we have also tried several pre-processing methods (tf-idf, feature selection, and centroid-based clustering) to improve the accuracy of the learning models. In the end, the best model is Gradient Boosting with Decision Tree using the pre-processed tf-idf data set.

The progress of the entire project has been smooth, though we did get some unexpected results: most of the data manipulation methods we have tried do not actually improve the performance of the models. For future improvement, more pre-processing methods should be explored.

*All authors contributed equally on this project and this report.

1. Data Manipulation

1.1. Selection of tf-idf

Tf-idf is a statistic used to measure how important a word is to a document in a collection of corpus. A naive method to measure this importance is word count which simply counts the number of times that term occurs in a document d. But tf-idf is more sophisticated since it also takes into account that whether the term is common or rare across all documents. Technically the basic ideas of tf-idf are: (a) if a word appears frequently in a document, then it is important, so we will give the word a high score; (b) But if a word appears in many documents, then it is not a unique identifier, so we will lower the score for the word.

Mathematically, tf-idf is calculated as

$$\text{tf-idf}(t, d, D) = tf(t, d) \cdot idf(t, D)$$

Here $tf(t, d)$ is the term frequency which is defined as the number of times that term occurs in a document d. Other methods to define tf(t,d) include the following[1]:

1. Boolean frequency: $tf(t, d) = 1$ if t occurs in d and 0 otherwise;
2. Logarithmic scaled frequency: $tf(t, d) = 1 + \log f(t, d)$, or zero if $f(t, d)$ is zero;
3. Augmented frequency:

$$\text{tf}(t, d) = 0.5 + \frac{0.5 \times \text{f}(t, d)}{\max\{\text{f}(w, d) : w \in d\}}.$$

$idf(t, D)$ is the inverse document frequency which measures how much information the word provides and is defined as

$$idf(t, D) = \log \frac{N}{1 + |\{d \in D : t \in d\}|},$$

where N is the total number of documents in the corpus and $|\{d \in D : t \in d\}|$ is the number of documents where the term t appears.

The reason why we also consider tf-idf is two-fold: Firstly, from its mathematical definition, this sophisticated method incorporates the commonness or the rarity of each word among all documents; In addition, empirically, through our implementation, we actually achieve higher accuracy and lower variance by applying tf-idf than word account for a given learning model (see Section 4.3 below).

1.2. Feature Selection

Since we have 500 features for each data point, then it is natural to think about a way to reduce the number of features, because this may potentially 1) make training faster and 2) avoid over-fitting on training data-sets. However, it is also possible to under-fit the data-sets, as we are artificially making the data-set to be less complicated. Therefore, in order to have a good balance of the effects of feature selection, cross-validation is crucial for feature selection.

For feature selection, we used the most common univariate feature selection, i.e. selecting the best features based on univariate statistical tests.

1.3. Centroid-Based Clustering

We also consider centroid-based clustering on the data-sets, i.e. by the euclidean distance of the feature vectors. Because for data-points with similar

feature vectors, they may intuitively have more similar correlation to financial recession. Hence, it is natural to first cluster the data-sets, and for each cluster, apply a different machine learning model. However, since we are dividing our large data-sets to a much smaller group, then for each group, we may potentially over-fit each one, since the data-set in each cluster may not be representative of the actual population.

For centroid-based clustering, we use the K-Means clustering, because it is currently the fastest clustering algorithm implemented.

2. Learning Algorithm

2.1. Basic Algorithms

2.1.1. Linear Regression

Since the problem is a binary classification, then it is natural to at least try to linearly separate the training data. Moreover, after reading similar research, we found out that many research groups have used SVM to predict the direction of the market[2]. Hence, we choose SVM as our learning model for linear regression in this project.

2.1.2. K Nearest Neighbors (KNN)

Similar to the choice of SVM, many research groups have used KNN for the prediction of market trend[2]. This algorithm basically classifies a new data point by using the neighboring training data points with some weights (e.g. the Euclidean distance between the neighbor and the new data). However, since KNN requires careful parameter choices to avoid over-fitting or under-fitting, and it is also a bit similar to clustering, then we only did some superficial implementation of this learning model for comparison.

2.1.3. Decision Tree (DT)

Decision Tree is another natural learning model for binary classification, besides linear regression. However, the problem with Decision-Tree only learning models is the necessity of choosing an appropriate stopping criteria. However, since our feature space is huge, parameter manipulation is very time consuming. Hence, naive application of Decision Tree will not give us high accuracy. In the end, for our implementation of Decision Tree, we always combine one ensemble algorithm to improve the performance of weak Decision Tree. In the next part, we will introduce the three ensemble algorithms we have used to improve the performance of Decision Tree.

2.2. Ensemble Algorithms

2.2.1. AdaBoost

AdaBoost is one of the most common ensemble algorithms. It basically combines several weak classifiers (e.g. a single split in Decisoin Tree) with different weights. One advantage of this algorithm is that it will improve the accuracy of weak learning models without over-fitting the data. Therefore, in our project, we combine Decision Tree with AdaBoost to train our learning model. The parameter we manipulated with AdaBoost is the number of estimators.

2.2.2. Bagging

Bagging classifier is a method for generating multiple versions of a weak or unstable classifier (or regression function) and using these to get an aggregated one. Let D be a training set of size n. Then bagging generates m bootstrap samples D_i of the training sets of size n' by sampling from D uni-

formly and with replacement. Then train a classifier or a regression function using each bootstrap sample. For classification, Bagging does the majority vote on the classification results, while, for regression, Bagging averages on the predicted values. We use Bagging to reduce variation and improve the performance for unstable classifier which varies significantly with small changes in the data set. The parameters we manipulated with Bagging are the number of estimators, the number of training points and the number of features we use.

2.2.3. Gradient Boosting (GB)

In addition to AdaBoost and Bagging, Gradient Boosting is the third ensemble algorithm that can be used on both classification and regression. It works like gradient descent in function space of weak classifiers, each classifier being a shallow Decision Tree in this case. The idea is to minimize the loss function $L(y, F(x))$ along an axis of the function space at each iteration. The parameters we manipulated with Gradient Boosting are the number of estimators, the learning rate, and the maximal depth of each estimator.

3. Model Selection

The way we compare different models is cross-validation. On our training-sets, we run 5-fold cross-validation with different learning models, and calculate the average accuracy and the standard deviation of the accuracy. For the optimal learning model, it should have a high average accuracy and a small standard deviation. For all the accuracy we are showing below, it is the measure of $mean \pm 2 \cdot std.dev.$ of the accuracy after 5-fold cross-validation.

3.1. Ensemble Methods with Decision Tree vs SVM vs KNN

In this section we compare the performance of our three basic models: Decision Tree (with ensemble method), SVM, and KNN.

1) For Decision Tree, we try it with AdaBoost in this section. We choose the number of estimators to be 10 and 50.

2) For SVM, we try it in two ways: regular SVM and SVM with AdaBoost. For SVM with AdaBoost we choose the number of estimators to be 10 or 50.

3) For KNN, we choose the number of nearest neighbors to be $5, 6 \cdots 9$.

Table 1: Accuracy (%) of AdaBoost with Decision Tree

Estimators	Accuracy of DT
10	85.35 ± 2.42
50	89.12 ± 2.22

Table 2: Accuracy (%) of SVM

Estimators	Accuracy of SVM
Regular	82.33 ± 0.43
10	82.33 ± 0.43
50	82.33 ± 0.43

From Table 1, Table 2, and Table 3 we see that ensemble method with Decision Tree outperforms KNN and SVM (whether with ensemble methods or not). The reason of the relatively low accuracy of KNN may lie in the distribution of the data: the distribution is not very smooth or the neighboring

Table 3: Accuracy (%) of KNN

K	Accuracy of KNN
5	83.75 ± 1.45
6	83.70 ± 1.46
7	83.75 ± 1.50
8	83.65 ± 1.50
9	83.60 ± 1.48

points influence each other through only part of all the features. Note that the number of estimators will not influence the performance of SVM. This makes sense, since ensemble methods deal with weak classifiers. For example, for Decision Tree, we can make weak classifiers by limiting the depth of each tree and the training process is stochastic. But with SVM, there is no difference between estimators we get from different iterations. That is the reason why SVM is not compatible with AdaBoost.

From this case, we can see that AdaBoost with Decision Tree can balance between over-fitting and under-fitting automatically. So in the following sections we will focus on ensemble methods with Decision Tree.

3.2. Different Ensemble Algorithms

Having chosen ensemble methods with Decision Tree as our main algorithms, in this section we will compare the performance between different ensemble methods.

First we want to determine the optimal parameters for the two ensemble methods, i.e. Bagging and Gradient Boosting.

1) For Bagging, we can add the following constraints to the algorithm: the number of samples to draw from X to train each base estimator ($MaxSamples$) and the number of features to draw from X to train each base estimator ($MaxFeatures$).

Table 4: Accuracy (%) of Bagging with Different Parameters

$Estimators$	$MaxSamples(\%)$	$MaxFeatures(\%)$	Accuracy of Bagging
50	1.0	1.0	91.90 ± 2.27
50	1.0	0.9	92.07 ± 2.19
50	0.9	0.9	92.32 ± 2.11
50	0.9	0.85	92.31 ± 2.27
300	1.0	1.0	92.18 ± 1.64
300	1.0	0.9	92.37 ± 1.89
300	0.9	0.9	92.43 ± 1.76
300	0.9	0.85	92.27 ± 1.79

From Table 4 we can see $MaxSamples = 0.9, MaxFeatures = 0.9$ is the best choice no matter the number of estimators. Hence, We will use these parameters in the future comparisons.

2) For Gradient Boosting we can choose different $LearningRate$ and the maximal depth of each decision tree ($MaxDepth$).

From Table 5, we can see it is best to choose $LearningRate = 0.1, MaxDepth = 4$ in the future comparisons.

3) Now we will compare all three ensemble methods with Decision Tree

Table 5: Accuracy (%) of Gradient Boosting with Different Parameters

$Estimators$	$LearningRate$	$MaxDepth$	Accuracy of GB
50	0.1	3	91.87 ± 1.87
50	0.1	4	92.35 ± 2.02
50	0.1	5	92.52 ± 2.18
50	0.1	6	92.37 ± 1.99
50	0.3	4	92.35 ± 2.24
1000	0.1	4	93.27 ± 1.57
1000	0.1	5	92.95 ± 1.89

Table 6: Accuracy (%) of Different Ensemble Methods

$Estimators$	AdaBoost	Bagging	GB
200	90.67 ± 1.85	92.24 ± 1.76	92.87 ± 2.00
500	91.83 ± 1.58	92.25 ± 1.64	93.17 ± 1.59
1000	91.85 ± 1.76	92.27 ± 1.60	93.27 ± 1.57

as the base model (with their correspoding optimal parameters). From Table 6, we see that Gradient Boosting outperforms AdaBoost and Bagging. Note that when the number of estimators is large, Gradient Boosting can achieve the highest cross-validation accuracy with rather low variances. The reason may come from the fact that Gradient Boosting is like gradient descent in function space. So when the number of estimators is large enough, it will stay around the optimal point stably.

Table 7: Accuracy (%) of GB with Different Number of Estimators

Estimators	Accuracy of GB
200	92.87 ± 2.00
500	93.17 ± 1.59
1000	93.27 ± 1.57
1200	93.27 ± 1.58

4) Moreover, we intend to manipulate number of estimators with Gradient Boosting a bit more to achieve even better accuracy. From Table 7, we see that when we run Gradient Boosting with 1000 estimators the accuracy is the highest and the variance is the lowest. If the depth of each estimator is too large, the efficiency of the algorithm will be impaired and over-fitting may occur.

Therefore, in the end, the optimal ensemble method we choose is Gradient Boosting with 1000 estimators, while the learning rate and maximal depth are 0.1 and 4.

3.3. Word Count vs tf-idf

Table 8: Accuracy (%) of word count vs tf-idf

	AdaBoost	Bagging	GB
word count	89.58 ± 2.52	91.62 ± 1.99	92.07 ± 2.42
td-idf	88.45 ± 1.45	92.05 ± 1.37	91.90 ± 1.80

To decide whether to use naive word count or tf-idf for our features, we have applied AdaBoost, Bagging, and Gradient Boosting to compare the traing of the two. By using 50 estimators for each ensemble method, we obtained the accuracy in Table 8.

From Table 8, we can see that for Adaboost and Gradient Boosting achieve similar accuracy in word count and tf-idf (though the accuracy of word count is slightly better) but the variance of accuracy for word count is larger than that for tf-idf. For bagging, td-idf has higher average accuracy (though slightly) and less variance, hence performing better than word count.

Since we are looking for learning models with more stability, then as the average accuracy between word count and tf-idf is similar across all ensemble methods, we then prefer to use tf-idf to word count, as the variance of accuracy is reduced.

3.4. Feature Selection

We apply feature selection to our training data and then train on specific models with the following steps:

1) Run a univariate feature selection algorithm that calculates the univariate statistics score on each feature, then we shrink the feature vectors by only choosing the highest p percentile features. We used $p = 65\%, 70\% \cdots 100\%$.

2) Then we apply a learning model to the transformed training data set for cross-validation. The learning models include AdaBoost, Bagging, and Gradient Boosting, which all have 50 estimators and Decision Tree as the base model.

From the cross-validation results (9), we can see that all feature selections have similar accuracy and standard deviation on the same model. Also

Table 9: Accuracy (%) of Feature Selection

p	AdaBoost	Bagging	GB
65%	88.90 ± 1.97	90.48 ± 1.97	91.55 ± 2.21
70%	88.40 ± 3.20	89.57 ± 2.60	91.57 ± 1.72
75%	87.55 ± 2.67	90.20 ± 2.64	91.80 ± 1.66
80%	88.92 ± 2.56	91.15 ± 2.42	92.10 ± 1.92
85%	88.50 ± 2.63	90.90 ± 1.10	91.80 ± 2.27
90%	88.12 ± 3.20	90.70 ± 2.15	92.02 ± 2.30
95%	88.78 ± 1.87	91.05 ± 2.40	92.00 ± 3.93
100%	89.30 ± 1.33	90.53 ± 2.27	92.20 ± 1.30

surprisingly, when using 100% of all the features, each learning model has a highest average accuracy and a lowest standard deviation. Hence, we cannot actually conclude that using feature selection can help improve our models. This may be due to the fact that some features with very low univariate statistic scores are essential to the classification.

3.5. Centroid-Based Clustering

The following is our implementation of getting the performance of centroid-based clustering:

1) Run K-Means clustering algorithm on our training set, where K is the number of returned clusters. The total number of training data points is N.

2) Group the training data into K clusters according the results given by K-Means. Assume for cluster i, it has N_i data points.

3) For each cluster i, run a 10-fold cross-validation by the given learning model. In our testing, we did Decision Tree algorithm with one of AdaBoost, Bagging, and Gradient Boosting, with 50 estimators. Then after the cross-validation, we have the average accuracy $\bar{A}_i$ and variance of accuracy σ_i^2.

4) Then for the overall performance of the given learning model is has the accuracy with

$$\text{mean: } \sum_{i=1}^{K} \frac{N_i}{N} \bar{A}_i, \text{ and standard deviation: } \sum_{i=1}^{K} \sqrt{\frac{N_i}{N}} \sigma_i$$

We ran the K-Means algorithm with $K = 2, 3 \cdots 10$ on each of the ensemble models (i.e. AdaBoost, Bagging, and Gradient Boosting), whose base model is Decision Tree. The results of accuracy are shown in Table 10.

Table 10: Accuracy (%) of Clustering

K	AdaBoost	Bagging	GB
2	89.13 ± 1.68	90.12 ± 2.59	91.67 ± 2.10
3	88.52 ± 2.69	90.05 ± 2.10	91.15 ± 3.29
4	88.48 ± 3.84	89.42 ± 2.59	90.75 ± 3.35
5	87.55 ± 3.94	89.37 ± 4.30	90.10 ± 3.48
6	87.15 ± 4.17	87.95 ± 3.37	90.57 ± 3.51
7	88.54 ± 4.73	87.96 ± 3.42	89.39 ± 3.88
8	88.07 ± 4.52	88.61 ± 4.50	89.36 ± 3.93
9	87.57 ± 4.72	87.56 ± 4.72	89.44 ± 4.49
10	87.53 ± 4.48	87.53 ± 4.48	88.96 ± 4.39

From Table 10, we can see that given a learning model (Adaboost, Bag-

ging, or Gradient Boosting), when we increase the number of clusters, the standard deviation of accuracy is also increasing. This is resulted from the the decreasing number of data-points within the each cluster, which is less representative of the data-points of this cluster (i.e. over-fitting the training-data). Moreover, the mean of accuracy is also mostly decreasing with much oscillations. Moreover, when we only have two clusters, the performance on each model does not perform as well as without clustering. Finally, interestingly, these results also confirm that Gradient Boosting outperforms Bagging, and AdaBoost is the least accurate learning model for our training set.

Therefore, either because our training-set is still too small to train each separate cluster, or because the entire population just does not have much clustering at all, centroid-based clustering is not going to improve our model's performance. In the end, we decided not to apply centroid-based clustering to our final learning model.

4. Conclusion

For data pre-processing, we have concluded that feature selection and centroid-clustering are not going to improve the learning model's accuracy. However, using tf-idf instead of the simple word counts will indeed improve the performance of our learning models by stabilizing the accuracy of our model.

As for specific learning models, we can see that ensemble methods with Decision Tree outperform both SVM and KNN. Moreover, among the three ensemble algorithms (AdaBoost, Bagging, and Gradient Boosting), Gradient Boosting is the most accurate, and AdaBoost is the least accurate model.

5. References

[1] Wikipedia, Tf–idf — wikipedia, the free encyclopedia (2015). [Online; accessed 26-February-2015].

[2] Y. Ko, A study of term weighting schemes using class information for text classification (2012) 1029–1030.