SI 670 Final Project: Predicting text difficulty

Jingyi Jia, Yixin Zheng, Shirley Wang

December 14, 2021

Abstract

In our project, we aim to extract out several useful features from the datasets to help to determine whether the sentence is difficult or not. We first used two simple classification models to set up a baseline for our model performance. Then we tried 6 more supervised learning methods from different families, such as the tree-based classifier, linear classifier, and nearest neighbor classifier. There are two types of data used either separately or together in different models: the textual features and the numerical features. We then used four different evaluation methods that are accuracy score, F1 score, precision score, and recall score to help to choose the best model. We mainly use the accuracy score as our primary method to evaluate the model performance. Since the natural language process field itself is still in progress to be learned and due to the limitation of our dataset, we found that the current model is not good enough to successfully classify every sentence into the right category. We may need more experiments and analysis to improve the model in the future.

1 Introduction & Motivation

Clarity is important while conveying complex ideas and concepts. However, in many real-world cases, people may face the problem that textual information is not comprehensible by audiences who may not have high reading proficiency. Machine learning methods give the possibility to classify text into difficult or easy in an efficient way. As a result, textual data analysis becomes an important but tough task in the machine learning field. It is hard to detect the difficulty of a single sentence due to the diversity of the text itself. In reality, there are lots of aspects of the text that can be analyzed, such as the part of speech of the word, the multiple meanings of the word used in different contexts, the familiarity of the word to a child, etc. Therefore, we need to find ways to simplify the complicated text, allowing people to easily understand and retrieve the most important information.

In this project, we aim to provide insights on whether the article's text needs to be simplified or not. In particular, we will classify each sentence into one of the two categories by assigning labels 0 and 1. In our case 0 represents the sentence that does not need to be simplified while 1 represents the sentence that does need to be simplified. However, the text content is hard to classify. To successfully achieve the goal, we need to consider different aspects of text such as the length of the sentence, the frequency rate of using the single word, the complexity of each word, etc. To accomplish this, we will extract some of the text features combined with numeric features from our dataset.

2 Data Source

2.1 Data Collection

We use the dataset provided in UMich SI 670 F21: Predicting text difficulty on Kaggle. The training dataset contains 416,768 sentences, labeled with one of the two categories (0: the sentence does not need to be simplified; 1: the sentence does need to be simplified.) The test dataset contains 119,092 comments that are unlabeled. We will need to classify them into two groups using the classifiers we build. The remaining three datasets are helpful to create the metric for measuring the difficulty of the text from different aspects. We hope to use some of the data columns from those three files to help to extract some valuable features from the training and test data to help to improve the classification result. We will mention more on how to do the feature extraction in section 2.2.

2.2 Data Preprocessing

In this section, we explore and extract important features of the data contained in the sentences by preprocessing the data. In particular, we extract two types of features from the original training and test dataset: text features and numerical features.

2.2.1 Text Feature

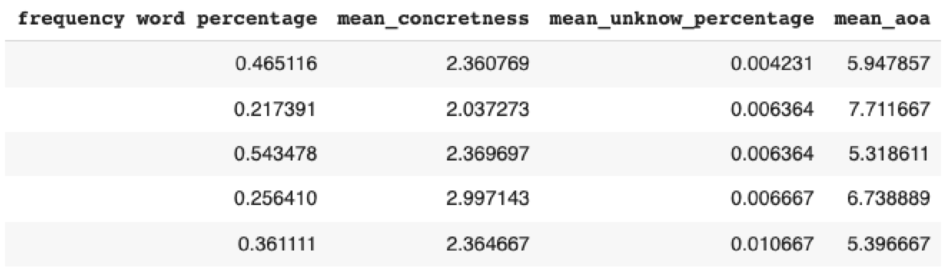
To extract text features, we first split up a larger body of text into words called tokens. Then we use word embedding to convert each token as a vector that represents the features of the token. We use these vectors as the features to represent the sentences. In our project, we first restrict the Wikipedia texts to the top 20,000 most common words and cut the reviews after only 40 words. In the later section, we use the LSTM neural network that learns 64-dimensional embeddings for each of the 20,000 words that turn the input integer sequences into embedded sequences and train multiple Dense layers for classification. In particular, we use ”pretrained word embeddings” which means the word embeddings were precomputed using a different machine learning task than the one we are trying to solve.

2.2.2 Numerical Feature

To extract numeric features from the training and test dataset, we combine the three files that measure the difficulty of the words with the training and test dataset to extract four important features (Figure. 1). They help to decide whether the single word or the entire sentence is complex or not. We achieve the first feature by calculating the percentage of frequently used words in each sentence. The words in dale_chall.txt are frequently used in real life and we may guess these words can be easily understood and be spoken by people. Therefore, we calculate the frequency of those words appearing in each sentence to represent the difficulty of a sentence. In other words, if one sentence consists of most of the simple words with a higher percentage of frequently used words, then the entire sentence will not be too difficult to understand.

The next two features we use are the mean concreteness and mean unknown percentage of each sentence. From the file named Concreteness_ratings_Brysbaert_et_al_BRM.txt, we observe that when the word has a high concreteness value, it is easy to understand. We achieve this result by randomly selecting the words from the file and classifying them manually. This file also provides us with the percentage of people who do not know the words. This is also a good feature to help to classify as a difficult word that may hardly be seen by most people. Therefore, we may reasonably guess that a high percentage of unknown rate implies the high difficulty of the word.

The last feature is to the mean AoA of each sentence generated from the file AoA_51715_words.csv. The AoA value represents the average age of people who first know the word. Since we believe that children always learn simple words at first, it is reasonable that the low value of this feature implies the low complexity of the corresponding word. The table in Figure 1 gives an overview of how these numerical features look like.
In our project, we use either text feature or four numeric features or both of them to train the different classifiers to make the classification.



frequency word percentage	mean_concretness	mean_unknow_percentage	mean_aoa
0.465116	2.360769	0.004231	5.947857
0.217391	2.037273	0.006364	7.711667
0.543478	2.369697	0.006364	5.318611
0.256410	2.997143	0.006667	6.738889
0.361111	2.364667	0.010667	5.396667

Figure 1: Numeric Features Example

3 Methods

We first fit two baseline models: (a) dummy classifier baseline and (b) a Naive Bayes classifier. We then fit several different supervised learning models including K nearest-neighbor, tree-based methods, LSTM, logistic regression, random forest, and Adaboosting. The LSTM models use both text features and numerical features while the other models use only numerical features.

3.1 Dummy Classifier

The dummy classifier is a baseline model used in the project to compare with other classifiers. Since the classifier uses the most simple rules to predict, we can use its performance as a baseline for other classifiers. In this model, we used the four numeric features to make classification and return the mean value of the evaluation score using 5-folds cross-validation.

3.2 Naive Bayes Classifier

Another baseline classifier we used is Naive Bayes which is a family of simple probabilistic classifiers. The features used in Naïve Bayes classifiers prefer to be linear since the classifier itself is highly scalable. We use the numeric features to train the classifier and predict the results for the validation set. The performance is calculated using the mean accuracy through 5-fold cross-validation.

3.3 K nearest-neighbor Classifier

The first supervised learning method we use is the k-nearest neighbors (KNN) algorithm. It is an easy-to-implement supervised machine learning algorithm that can be used to solve classification problems. In our case, we first load the numerical features of the training dataset. We then use 5-fold cross-validation to train and evaluate the classifier. In each run, we normalize the feature so that the scale of different features does not affect the result. To avoid data leakage, we normalize the training set first and then normalize the validation set using the same scaling of the training data. Then we initialize K = 1 as the number of neighbors. We set K = 1 because after plotting the accuracy vs. k value curve and we notice that the accuracy keeps decreasing as k increases and K = 1 gives the highest accuracy. This, however, will be likely to cause overfitting problem which we will discuss in the later section. We then build a KNN classifier using the optimal K. We use 5-fold cross-validation with accuracy, f1 score, precision, and recall as the metric to evaluate the performance of the KNN classifier. We will compare the results of the KNN classifier with other classifiers in the later section.

3.4 Decision Tree Classifier

Secondly, we use the decision tree classifier to predict the difficulty of the sentences. The decision tree algorithm selects the best attribute and by making that attribute a decision node, it splits the data set into smaller subsets. The tree can thus be built by repeating this process recursively until there are no more remaining attributes. In our case, we use 5-fold cross-validation to train and evaluate the performance of the decision tree classifier. That is, in each of the 5 runs, we first split the data into the training set and validation set. Then we use their numerical features as the attributes to construct the tree using the training data and then use the validation set to check the performance of the classifier using the mean of accuracy, f1 score, precision, and recall as the metric. The decision tree algorithm does not require normalizing the feature due to its property. Furthermore, compared to a black-box type of algorithms such as Neural Network, decision tree displays internal decision-making logic. Therefore, we can display the graph of the importance of features to show the most important feature used to classify the data set. From Figure 2, we can see that "mean_aoa" is the most important feature in classifying the data set. On the other hand, "mean_unknow_percentage" is the least important feature in classifying the data set.

3.5 Random Forest

We applied another tree-based algorithm, random forest, to improve the model performance. Compared with the decision tree, which uses only one tree, random forest is an ensemble of many decision trees. Firstly, it randomizes bootstrap copies from the original dataset. Then it randomly split the features, looking at a random subset of possible features. Finally, it combines the predictions. The input features are the numeric features. We used 5-fold cross-validation to tune the model. It keeps the same order or feature importance (Figure 3) with the decision tree.

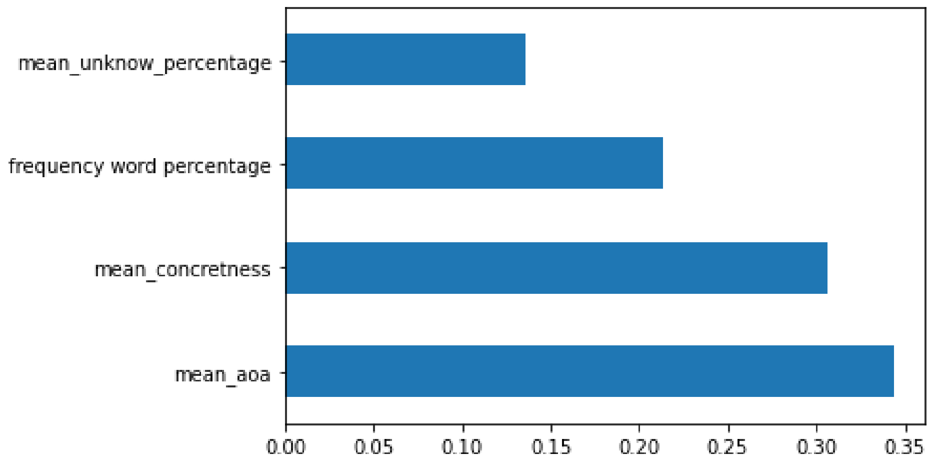


Figure 2: Feature importance of decision tree classifier

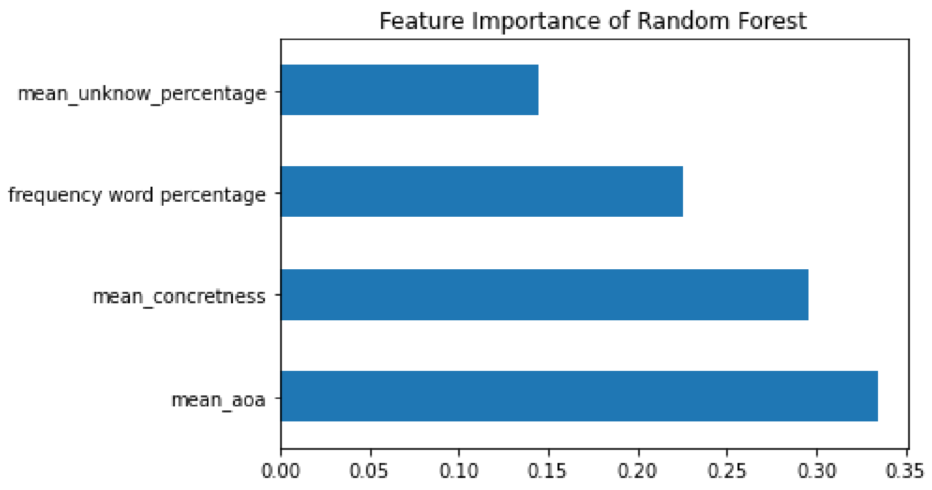


Figure 3: Feature importance of Random Forest classifier

3.6 AdaBoost

We used AdaBoost as another classifier. Similar to the random forest classifier, it is also a tree-based ensemble algorithm. The difference is that AdaBoost gives different weights to samples. Specifically, it gives higher weights for wrongly predicted samples. We use numeric features of the training data to train the classifier and do the prediction. We can plot the importance of the feature (Figure reffig:ad$_fi).Wenoticethat$“$frequencywordpercentage$”$isnolongerthemostimportan$

3.7 XGBoost

We used another tree-based ensemble method, XGBoost to train our model. It is similar to gradient boosting but regularizes the tree weights. It keeps the same order of feature importance as the previous tree-based algorithms(Figure 5).

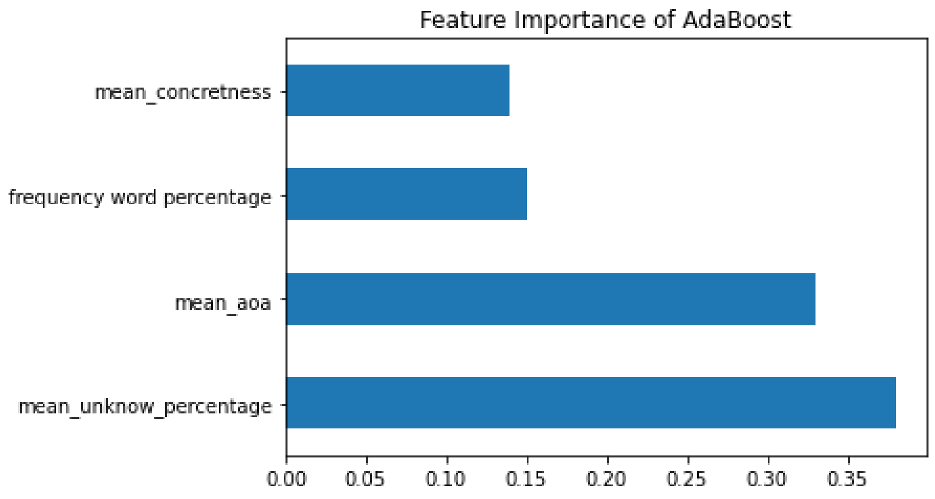


Figure 4: Feature importance of AdaBoost classifier

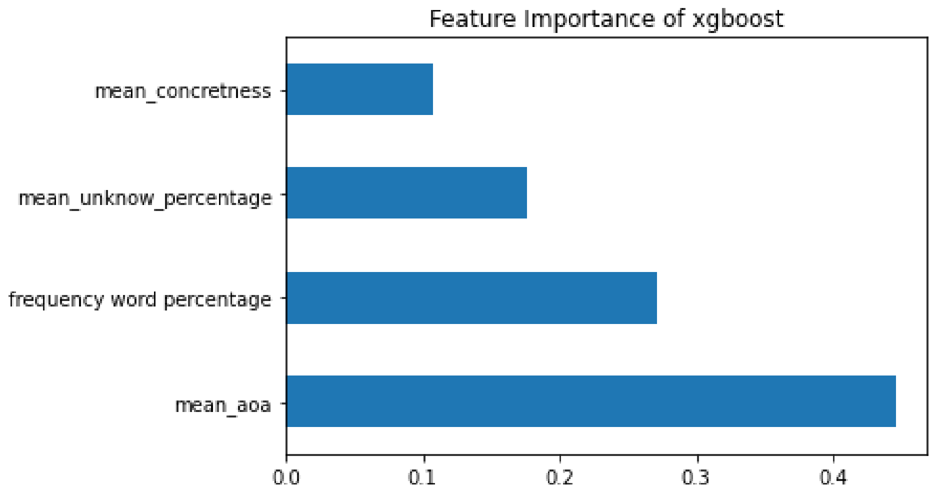


Figure 5: Feature importance of XGBoost classifier

3.8 Logistic Regression

Logistic regression is another classification algorithm used in our project. After splitting data into a training dataset and validation dataset, we perform the data scaling using the StandardScaler function. We can find the feature importance according to the coefficient value. The result shows us that the AOA value is most important (Figure 6).

3.9 LSTM

We use Long-short term memory(LSTM) in the deep learning field as one of our models. Unlike the previous models that only use numerical features, we train this model by including different features. We use 5-folds cross-validation to split the data and train the model. Before training the model, it is important to scale the numerical feature data.

In the first case, we only include the text feature to make the classification. To improve

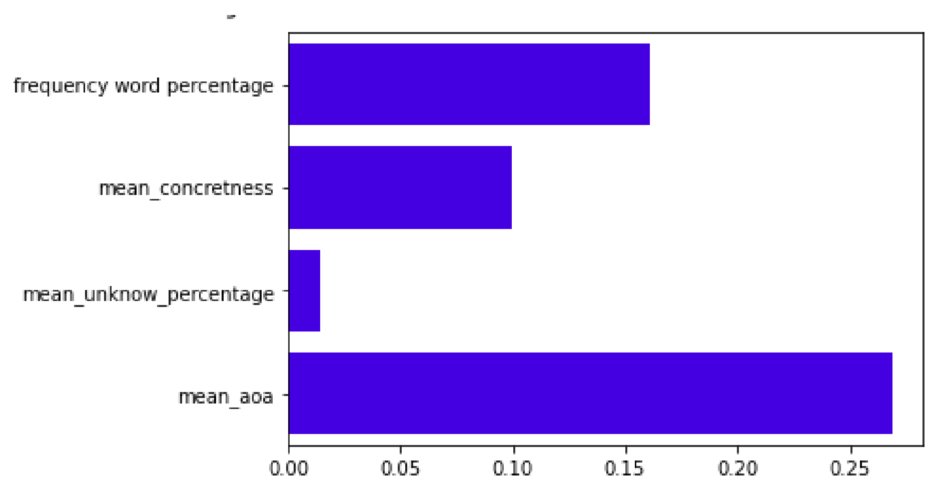
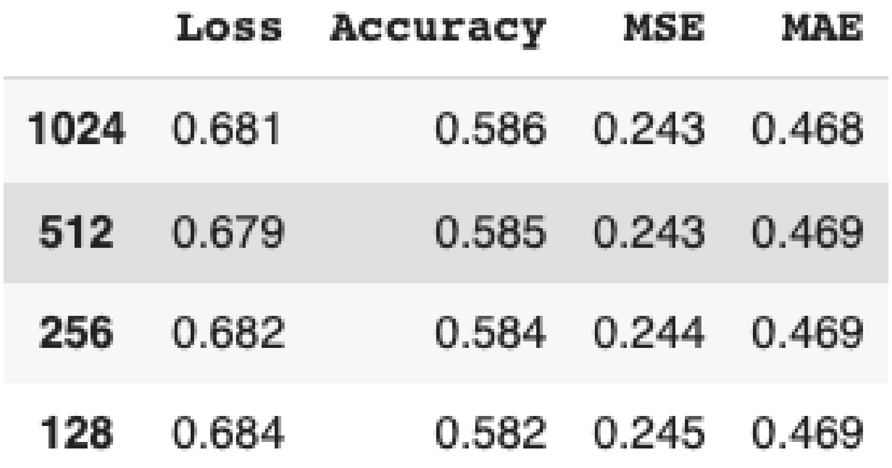


Figure 6: Feature Importance for Logistic Regression

the performance of the model, we try to tune the batch size parameter and to choose the best parameter according to the mean accuracy score return after the 5-folds cross-validation process. We also calculate loss function score, MSE and MAE to help to evaluate(Figure 7).



	Loss	Accuracy	MSE	MAE
1024	0.681	0.586	0.243	0.468
512	0.679	0.585	0.243	0.469
256	0.682	0.584	0.244	0.469
128	0.684	0.582	0.245	0.469

Figure 7: Evaluation Score for LSTM

In the second case, we try to combine the text feature with four numeric features into our LSTM model. In each fold, we tried two different scalers which are MinMaxScaler and the StandardScaler. After observing the accuracy of the validation dataset, we decide to use the StandardScaler in each fold. To combine the text and numerical features, we need to use the end-to-end model to train two types of data simultaneously by the same target label. In this case, we need to build two input layers separately for our NLP features and our numeric data. That is, we use the bidirectional LSTM model to deal with the NLP feature and then combine the output with the numerical features. We meet a problem related to the size of the datasets and we will talk more about it in the failure analysis section.

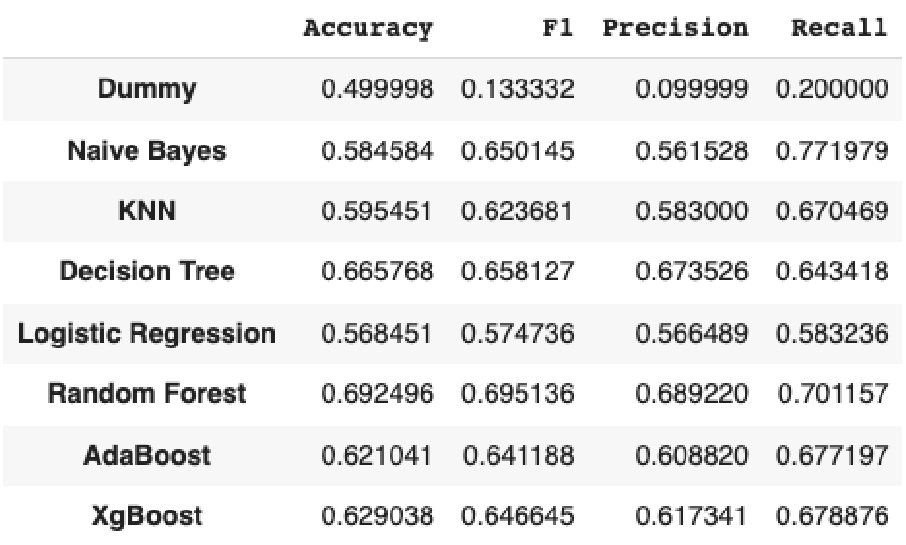
4 Evaluation

4.1 Evaluation Metrics

We use 5-fold cross-validation to compare the performance of the above three different supervised learning methods.

4.2 Main result

From Figure 8, we can see Random Forest has the best performance in accuracy, F1 score, precision, while Naive Bayes has the best performance in Recall. We apply 5-fold cross-validation to every model, and below are scores from that. Finally, we decide to use a Random Forest as our best model. All the models perform better than the baseline model dummy model.



	Accuracy	F1	Precision	Recall
Dummy	0.499998	0.133332	0.099999	0.200000
Naive Bayes	0.584584	0.650145	0.561528	0.771979
KNN	0.595451	0.623681	0.583000	0.670469
Decision Tree	0.665768	0.658127	0.673526	0.643418
Logistic Regression	0.568451	0.574736	0.566489	0.583236
Random Forest	0.692496	0.695136	0.689220	0.701157
AdaBoost	0.621041	0.641188	0.608820	0.677197
XgBoost	0.629038	0.646645	0.617341	0.678876

Figure 8: Evaluation Score for Models

4.3 TradeOffs

In random forest, there is a tradeoff between false positive rate and false-positive rate. By predicting probability and calibrating the threshold, we can balance those two concerns in the model. We plot the ROC curve(Figure. 9) of the random forest to give a direct view.

In decision tree method, we plot the precision-recall curve to visualize the trade-off between precision and recall (Figure 10). We can see that as precision increases, recall decreases, showing a tradeoff between precision and recall.

4.4 Sensitivity Analysis

The default parameters work well in Random Forest. According to Figure 11, When "max depth" is over 30, it reaches the best performance and becomes stable; when it is less than 30, the performance gets worse when reducing the "max depth".

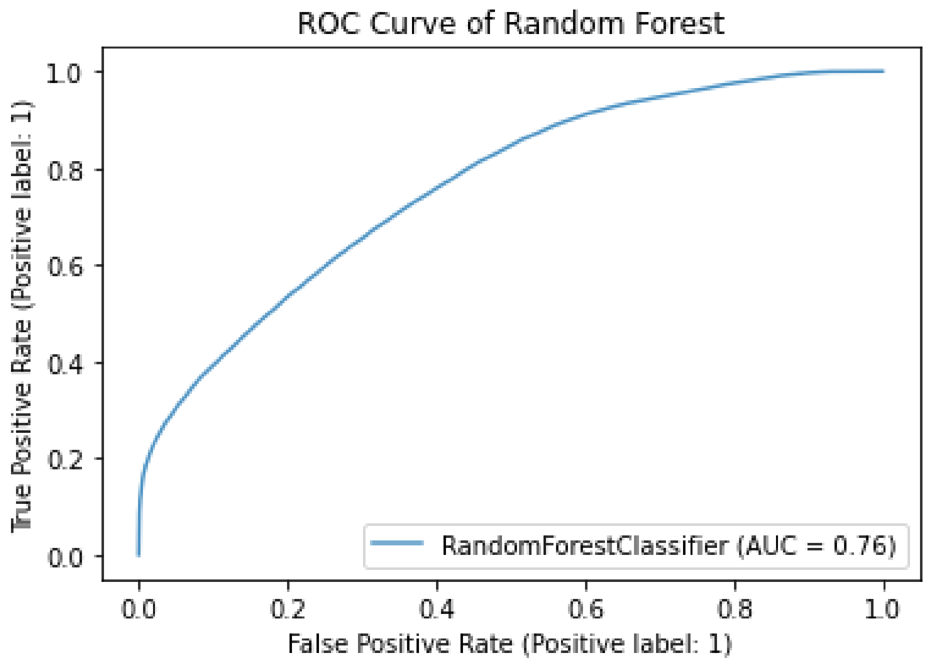


Figure 9: ROC for Random Forest

4.5 Feature Importance

According to Figure 3, in our best model Random Forest, “Mean AoA” is the most important feature, while “mean unknown percentage” is the least important feature. “AoA” is the age of acquisition, which refers to the approximate age (in years) when a word was learned.

4.6 Failure Analysis

One of the difficulties we encounter in the project is building the LSTM model using both the text features and numeric features. The problem here is that when we input the NLP features into the neural layer, the size of the normalized textual dataset is too large to be trained using the local system. We can hardly find a better way to deal with this problem and decide to manually split the data into small datasets. However, the problem here will be related to the data leakage. Since the text data is not linear continuous, we can hardly get a good performance when using 200 sentences as our training data to predict our test data.

5 Discussion

From our analysis, we notice that both numerical features and text features extracted from the data can be used to classify the data set. Both the decision tree classifier and random forest classifier display the “mean_aoa” to be the most important feature and “mean_concretness” to be the least important feature to classify the data. However, the random forest classifier gives a better prediction result than the decision tree classifier. This improvement makes sense because the random forest is an improvement of decision trees that can limit overfitting without substantially increasing error due to bias. On the other hand, methods such as KNN and logistic regression do not work well compared to other classifiers with an accuracy lower

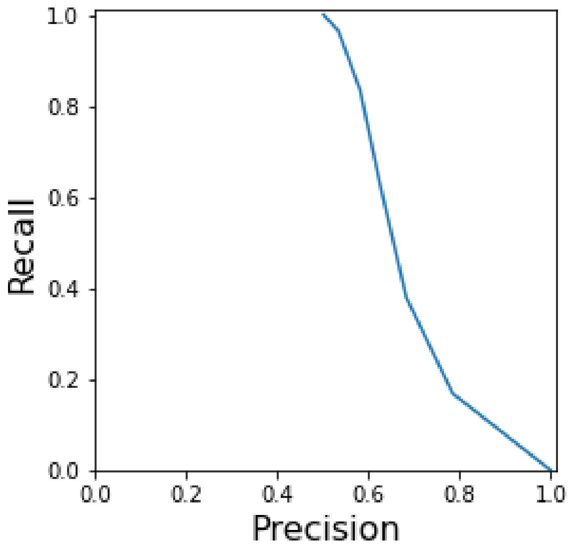


Figure 10: Precision Recall curve for KNN classifier

than 0.6. Logistics regression does not work well may due to the reason that the relation between the dependent variable and the independent variables may not be linear so that the assumption of the model does not meet. One possible reason that KNN does not work well is overfitting. Even though we find that the optimal k is 1, setting k to be 1 can easily make the model overfit; thus leading to a poor result on the validation set.

We are surprised to see that even though LSTM is the most complex model, it does not give the best results. This may be due to the reason that LSTM includes too many parameters that lead to an overfitting problem. As we run more epochs, the training accuracy increases while the test accuracy decreases, suggesting that overfitting exists.

We can extend our solution with resources on evaluating words difficulty. So far, we have some documents that evaluate the difficulty of the words using different criteria. In this project, we use them to generate numerical features for each text. If we are given more such documents, we will be able to create more numerical features; thus describing the text characteristics more comprehensively.

Even though it is attempting to use machine learning methods to classify the difficulty of texts, we cannot entirely rely on artificial intelligence and ignore the actual needs of people. Since there are people with different backgrounds and the ability to use English, we need to take into consideration of their opinions as well.

6 Statement of Work

Jingyi Jia: data preprocessing for text feature, knn, decision tree
Yixin Zheng: Random Forest(final model), AdaBoost, XGBoost, final evaluation
Shirley Wang: data preprocessing, baseline model, LSTM, Logistic Regression

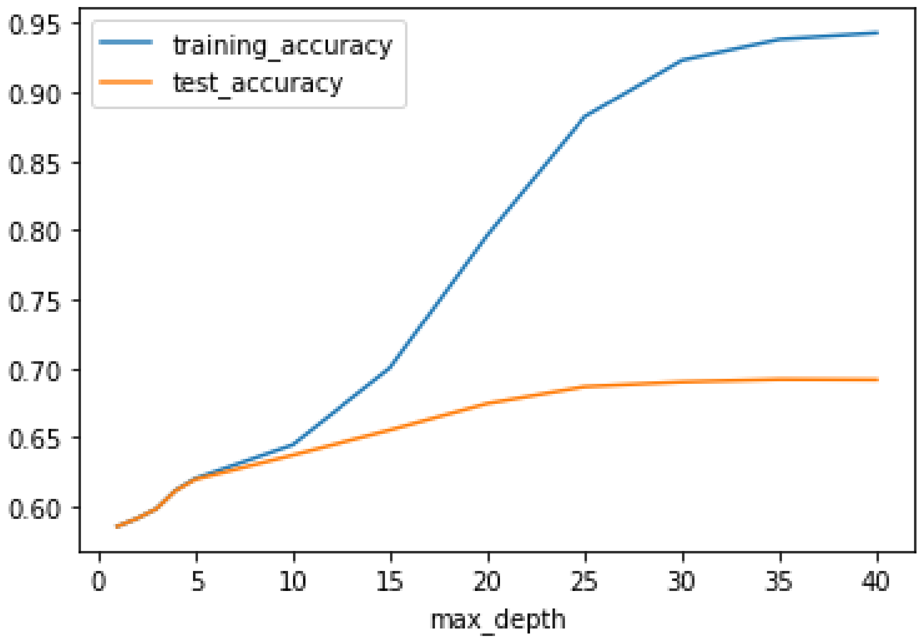


Figure 11: sensitivity of max depth in Random Forest