Predicting Text Difficulty

Biswajit Kumar| Pankaj Kumar Singh| Pierre LeBlanc

Problem Definition: Predicting Text Difficulty

Many applications exist for simplifying text. One application could be if a teacher would like to know if a textbook is suitable for a given student's reading level before proposing the book as homework. Our goal in this paper is to present analysis on whether the text needs to be simplified or not. We look at different supervised and unsupervised methods which help the text difficulty prediction task. Further detail on the problem is described in the Kaggle competition link for the readers reference.

Part A. Supervised Learning

Motivation

Manually reviewing sentences from a corpus and classifying it to simple and not simple is a tedious and labor-intensive work. Text classification takes the word vector form of the text as inputs and predicts the target classification label for that text. We aim to use supervised machine learning for text classification of a given sentence(s) as simple or not simple sentence(s).
The training dataset provided consists of text (primarily English text with some other non-English text) and labeled as 0 for simple and 1 for not simple sentences from a readability complexity perspective. We will use this data for training different machine learning models and choose the best accuracy model for prediction on unseen test data provided with text and without the label. This will allow editors to spend less time in simplifying the text and increase its readability.

Data sources

Data set

Location: Michigan 696 MADS course default data set:
https://www.kaggle.com/competitions/umich-siads-696-f22-predicting-text-difficulty/data

Description: Provided in Appendix - section 4

Methods & Evaluation

Feature extraction and representation

We begin our work first by manual review of the datasets to get a sense of the dataset's columns and purpose, whether the dataset is balanced or not, missing values, percentage and how best features can be extracted for modeling purposes. A lot of time is spent during preprocessing, and we have applied the ones which are frequently used in NLP pipelines like Lower casing, Removing special characters, Removing stopwords, Sentence Tokenization, and Words Lemmatization. The purpose was to perform data cleaning and analysis, build new features from the existing dataset and assess the feature impact on model performance. Given the problem is text classification, we tried different approach to convert text into different feature vectors eg. **Countvectorizer**, **TF-IDF** (**T**erm **F**requency-**I**nverse **D**ocument **F**requency) and **Doc2Vec**. We extracted 20 features (listed in Appendix – 2.a) based on datasets provided and narrowed down on thirteen features (listed in Appendix- 2.b) through feature ablation to build supervised and unsupervised learning models for the binary classification prediction problem. We

converted the entire dataset of features into vectorized form readable by machine learning algorithm for text classifications.

Model Selections

The problem in hand is a binary text classification problem and hence Supervised learning classification models are assessed to predict the class labels. After creating a 80:20 train-test-split of the training datasets, we first applied Dummy classifier (which predicts without trying to find patterns in the data) with parameter strategy as {"most_frequent" and "uniform"}, to serve as baseline model and compare with other models. For Linear models, we applied Logistic Regression as a classifier which uses the weighted combination of the input features and passes them through a sigmoid function. Sigmoid function uses maximum likelihood estimation approach to output a probability between zero and one which in this case would work for assigning text to target label 0 for simple and 1 for not simple based on a threshold value for the classifier. We applied a Logistic Regression classifier on the TF-IDF feature to compare the accuracy score. Next, we selected Naive Bayes (BernoulliNB) classifiers which tend to be faster in training and look at each feature individually to collect simple per class statistics from each feature. BernoulliNB is mostly used in text data classification and counts how often every feature of each class is not zero.

The next two models (RandomForest and XGBoost) that we implemented are part of ensemble decision tree classifiers which combine multiple machine learning models to create powerful models. These classifiers would not need to depend on spatial relations but instead on hierarchies of partitions among categories. The random forest classifier would avoid overfitting (a common problem with decision trees) by using different randomly selected samples and randomly selected features when training the trees. And the gradient boosted decision tree classifier would use the power of an ensemble of light-weight shallow trees, each of which could correct the mistakes of its predecessors

Third model applied is from the class of Neural Networks. Deep learning architectures are able to achieve high accuracy scores for text classification because they are able to learn features given enough training data. We found that the Multilayer Perceptron (MLP) provides competitive scores with only the ~half million training sentences and thus triggered curiosity within the team to use pre-trained models like BERT for the given problem. Please see our assessment on BERT in the sections below.

Base models	Dummy classifier	Decision Tree	Naïve Bayes	Gradient Boosting	Logistic Regression	RandomForest	NN MLP
Accuracy Score	0.49	0.62	0.66	0.67	0.68	0.69	0.70

Table 1: Summarizing the accuracy score for all models assessed

Hyper-Parameter Tuning :

To find the best parameters for the model we applied hyper parameter tuning in order to improve the performance. By using GridSearchCV we were able to try different combinations of values to find the model with the lowest error metric. Building the RandomForest classifier with different regularization parameters using GridSearchCV, we've succeeded to improve the accuracy scores to 0.71. This was the time consuming and critical step to find out the best parameters and re-train our classifier based on best parameters. It took more than 4 hours to run GridSearchCV and select the parameters. The parameters selected to hypertune for the classifiers are : criterion': ['gini', 'entropy'],'bootstrap': [True],'max_features': ['auto', 'sqrt'], 'n_estimators': [100,200]. Based on our sensitivity analysis we allowed the model to select the max_depth.

The best values for each parameter after GridSearchCV are :

```
(n_estimators=200, bootstrap= True, criterion='entropy', max_depth = None, max_features='sqrt', n_jobs=-1, random_state=42)
```

Feature Importance

We wanted to understand the top ten features which are contributing to the model performance. Below graph shows the top ten features and we learned that the words like ”lrb” and “rrb” which are HTML tags and other words are “stop words” and hence we assume that the “stop words” may have some impacts to the model which could be further investigated.

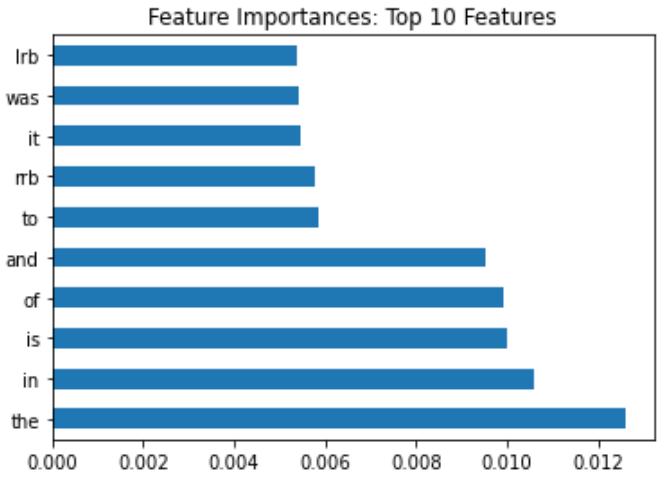


Fig 1: Top 10 features of Random Forest Classifier

Model Evaluation and performance

The selected random forest classifier went through the following evaluation pipelines.

- Split the training data into train and test using 80:20 rule
- Fit the classifier using the vectorized (TF-IDF) training data
- Make predictions using the test data
- Print a classification report based on the test data target label and predictions
- Plot a confusion matrix based on the test data target label and predictions

	precision	recall	f1-score	support
0	0.72	0.69	0.71	41525
1	0.71	0.74	0.72	41794
accuracy			0.71	83319
macro avg	0.71	0.71	0.71	83319
weighted avg	0.71	0.71	0.71	83319

Fig 2: Classification report for RandomForest Classifier

```
[[28723 12802]
 [11053 30741]]
0.7136907548098272
```

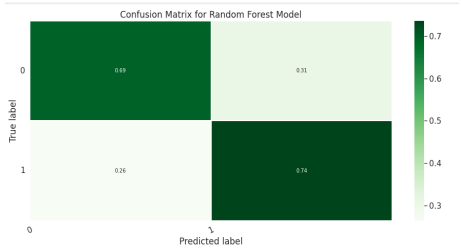


Fig 3: Confusion Matrix for the Classifier

We can see that precision and recall for both the classes do not have huge differences with F1 score of 71%. Model correctly predicted 28723 texts as simple, and 30741 as not simple (69% and 74% respectively) in the predictions. Model predicted 12802 as simple though they are not simple sentences (False Positive), and 11053 as not simple though they are simple sentences (False Negative).

Precision- Recall and ROC (Receiver Characteristic Operator) - Curve

ROC Curves and Precision-Recall Curves provide a diagnostic tool for binary classification models. From the precision-recall curve, we can observe the tradeoff between precision-recall represented by high precision and high recall with AP = 81%. The classifier scored more than 71% with ROC-AUC equal to 80%. Based on the AUC value the classifiers performance is acceptable.

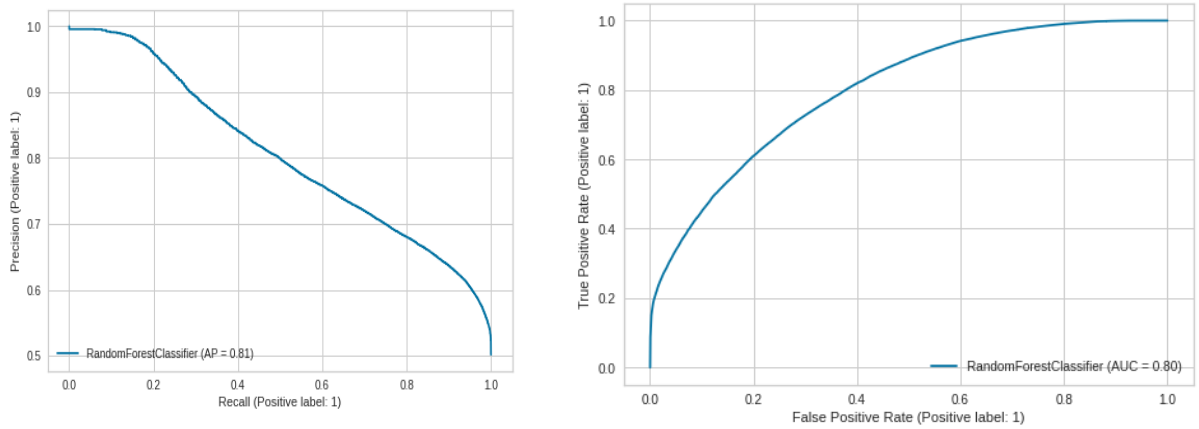


Fig 4: Precision-Recall and ROC-AUC curve for RandomForest Classifier

Failure analysis

To check how the classifier is failing, we performed a failure analysis by creating a confusion matrix and analyzed the specific failure instances using LIME.

LIME based Failure Analysis

We used Local Interpretable Model-Agnostic Explanations (LIME) for understanding the local model interpretability for Logistic Regression, Random Forest and BERT. LIME modifies a single data sample by tweaking the feature values and observing the resulting impact on the output. The key thing we wanted to know is, why was this prediction made or which variables caused the prediction?. The output from LIME below reflects the contribution of top-6 features towards the model predictions.

Logistic Regression: LIME predictions example

In the input text below, the model has classified the sentence as not simple (label=1) though the ground truth is opposite (label=0), we can say that model learning of the words like “form”, “separate” as being not simple is the reason for this prediction and needs to be further explained.

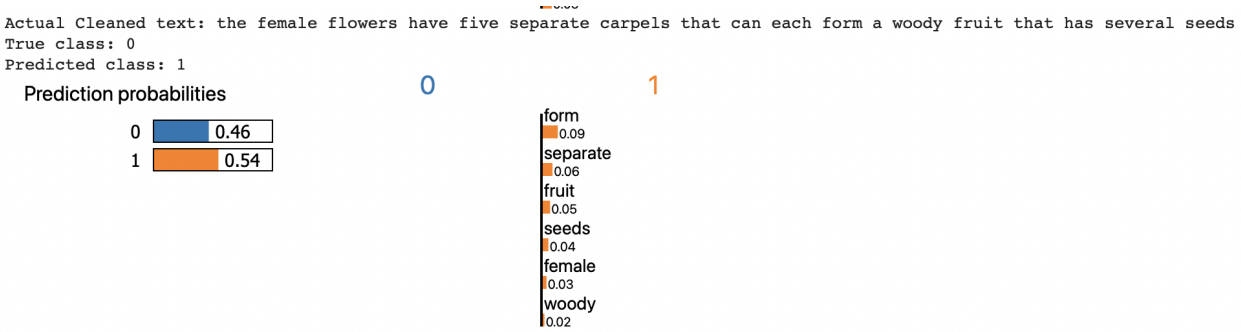


Fig 5: Logistic Regression LIME Prediction Analysis

Random Forest: LIME predictions example

In the input text below, the model has classified the sentence as not simple (label=1) though the ground truth is opposite (label=0), we can say that model unfamiliarity with the words like “rrb”, “lrb” could be the reason for predicting this sentence as not simple.

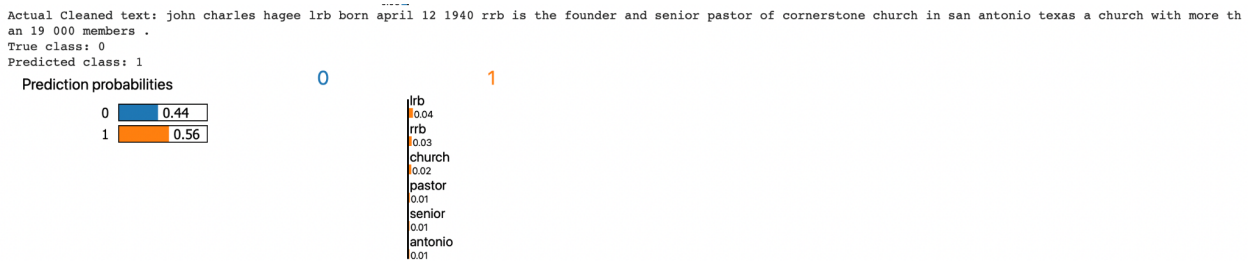


Fig 6: Random Forest LIME Prediction Analysis

Pretrained BERT: LIME predictions example

In the text below, the model has classified the sentence as simple/easy (label = 0), however the ground truth label is difficult (label = 1). Could be in this case the word “duffel” is not clearly understood in the context of BERT pretrained embeddings. Duffel in this case should be a difficult word in the broader context in the sentence and not a duffel bag, for example. More research is needed to see if proper nouns, places (municipalities) and locations are part of the BERT vocabulary.

Original sentence in lowercase: “*duffel is a municipality in the belgian province of antwerp*”

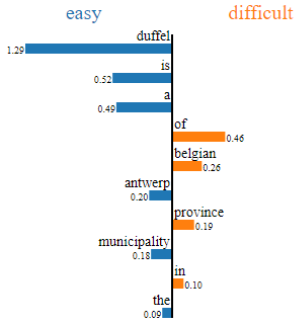
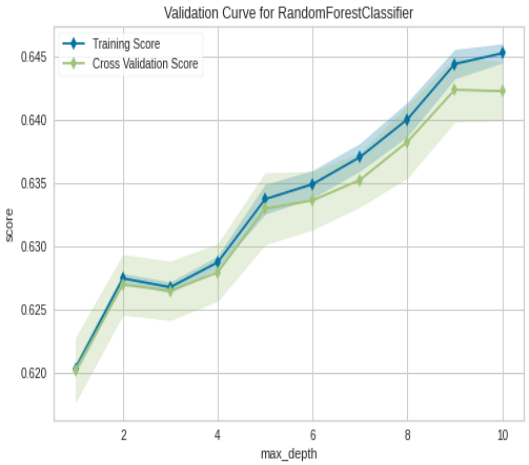


Fig 7: pretrained BERT LIME Prediction Analysis

Sensitivity Analysis:

We performed sensitivity analysis with the best extracted parameters to understand model variability and generalizability ability. We chose **max_depth**, **n_estimators** as parameters for sensitivity analysis and plotted against the accuracy at various values. We observe that at the max depth above 9 the score drops but overall, the model fits optimally at max depth below 9. Based on this, we let the model choose the max depth for better score and avoid overfit or underfit. Also, while plotting the hyperparameter **n_estimators** against the score we observe that the score slightly increases at the value 300 with no change in training score.



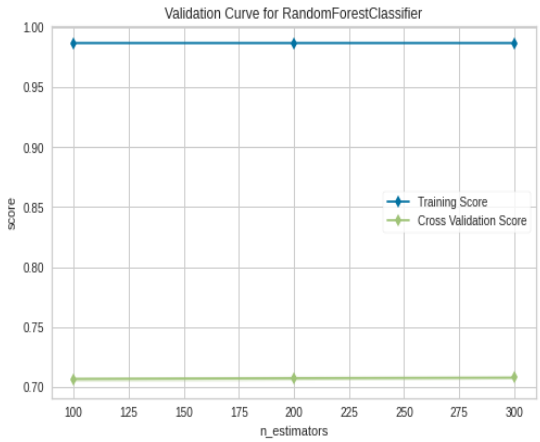


Fig 8: Sensitivity analysis with parameter max_depth

Fig 9: Sensitivity analysis with parameter n_estimators

Learning Curve :

Plotting amounts of training data (x-axis) against an evaluation metric, for multiple training set sizes and understanding if more data would help with respect to the overall evaluation metric. This also allows us to understand how the variability of classifier predictions is sensitive to the amount of training data. Though we consider split datasets but can observe that with increase in data set size the score increases. We can assume that with more data feeding to the classifier the performance will probably increase.

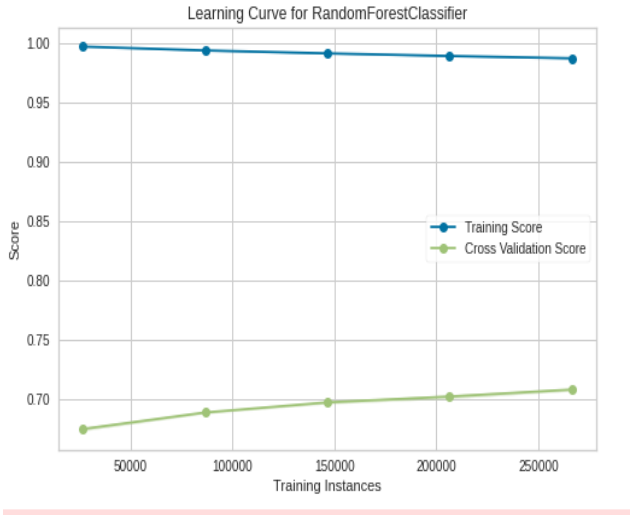


Fig 10: Learning Curve for Random Classifier

BERT PRE-TRAINED MODEL

We used BERT pretrained models from hugging face to improve the overall performance of our model and we wanted to explore advanced models within NLP. BERT is one of the more recent models within NLP to help with text classification and other language tasks such as question answering, abstract text summarization, translation, and other tasks. **BERT** stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. To understand BERT it is helpful to understand what is underneath BERT as shown

in the diagram below. We found it extremely helpful to review the videos from Chris McCormick where he takes us through the explanation of BERT. (Source: [3]) See graph below.

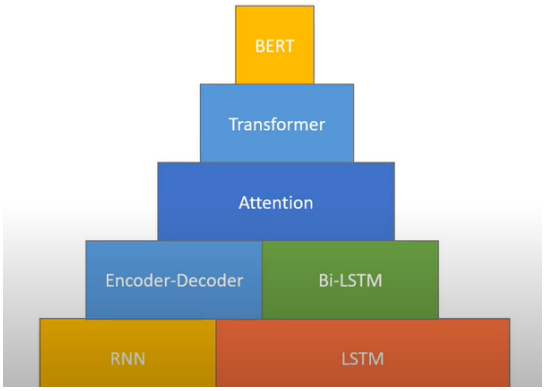


Fig 11 : BERT model source Chris McCormick [3]

BERT can contextualize each word for each sentence based on the surrounding words. This differs greatly from Word2Vec where there is only one fixed word embedding per word. The BERT base model has 12 transformer/encoder layers and 110 million trainable parameters whereas the BERT large model has 24 transformer/encoder layers and 340 million trainable parameters.

We reused and updated the code from Chris McCormick (source [4]) and modified the hyperparameters and also fine-tuned using our training data set "WikiLarge_Train.csv" from Kaggle ~half million sets of sentences. Here are the parameters we tried:

1. BERT Large, BERT base pretrained models
2. changing batch size: 32, 64, and 128
3. changing sentence length: 64 and 128
4. changing number of epochs (1 - 9)
5. changing drop out rate (1%, 5%, 10%, 50%, 60%)
6. removing punctuation vs. not removing punctuation
7. running full training set ~half million records vs. smaller training sets such as random 50k or 100k sample of sentences

All tests above were performed on the Michigan Great Lakes platform using the Nvidia Tesla V100-PCIE-16GB GPU on top of a multicore CPU and 16GB RAM. We thank the University of Michigan for allowing us to run all these tests of the BERT pretrained model. (Source: Great Lakes [5])

BERT results:

We tried many different parameters listed above. In general the BERT "large" was able to outperform by about 1 or 2 percentage points of accuracy compared to the BERT "base" version. Batch size 128 and sentence length 128 were not able to fit on the single GPU and ran out of memory. However, we found that 64 batch size and sentence length performs slightly better compared to 32 batch/sentence length. After trying the multilingual vs. the regular BERT we found there is not much difference in accuracy score overall. We were hoping for model improvement since there are a certain percentage of sentences and words which are not English within the training data set. The parameter which does have a dramatic effect on the overall score is "drop out". Dropping out of 50% or more probability has a dramatic effect on the model performance in a negative way. Drop out decreases the number of parameters and in this case 50% or more generates an overly simple model which underfits the data and the model goes significantly down in accuracy score. In many cases by increasing the drop out rate to this value of 50%

or above we saw a score that was similar to the dummy classifier which is equivalent to random guessing.

The overall best score we were able to get was using a drop out rate of 1% and batch/sentence length of 64 using the “bert large” version. Our score on Kaggle for that set of parameters resulted in 80.949% which is very close to 81% score.

Future improvement of this score using BERT would need to utilize BERT large, a batch size/sentence length of 128 or greater and also be able to utilize more than 1 GPU. We hope to explore these settings in milestone 3.

Here are some results from randomly selecting 100,000 sentences from the total ~half million training data set varying drop out and keeping the other parameters fixed using the BERT base version. Within this selection of 100,000 sentences we chose 50% training and 50% validation. One can clearly see the drop out of 50% degrades the validation accuracy (red line) significantly from ~70% to ~58%.

Drop out = 50%

Drop out = 10%

Fig 12 : BERT validation performance on 100,000 random sentences

Part B. Unsupervised Learning

Motivation

Same as above data source in Part A Supervised section

Data Source:

Same as above data source in Part A Supervised section

Methods & Evaluation

Principal Component Analysis (PCA):

PCA background:

PCA is an unsupervised dimensionality reduction algorithm. (source: 2) We used TF-IDF to transform our text to features within our supervised section of solving this prediction text difficulty problem. Within NLP and using the TF-IDF vectorizer this results in thousands of columns as features which represent each unique word in the vocabulary within the training data set. The purpose of PCA is to reduce the number of dimensions and in our case we are reducing from thousands of features down to two.

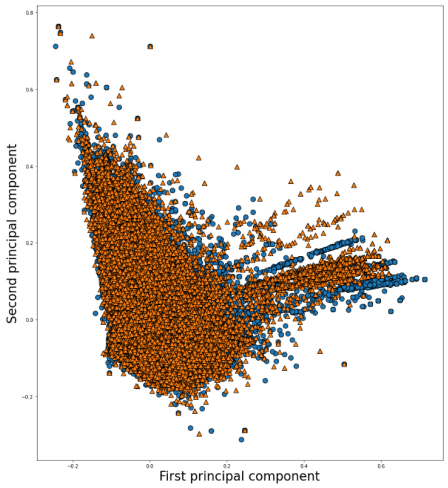
PCA results:

We used PCA to visualize the high dimensional data in two dimensions to see if there were any patterns in the training data set of sentences provided by Kaggle. Before starting any analysis we had a hypothesis there could be French sentences mixed in the data set, for example, and those sentences should show up clustered differently in a two dimensional space compared to the rest of the training data set of English sentences.

We ran PCA using the sklearn library with 2 components. Both regular PCA and Kernel PCA with radial basis function (rbf) were evaluated. Kernel PCA resulted in similar graphs and are not shown below as they are a duplication of the same.

The class separation within PCA is not good overall. However, there are clear patterns in the right part of the graph which show formations of lines skewing off to the right. After reviewing multiple samples of these points they do appear to be sentences which are in different languages (french, german, italian and spanish). However, it is not clear for the other part of the graph where the classes are directly on top of each other if there are also mixed languages in those sentences as well. The purpose of the PCA visualization is to see if there are any patterns within the data and if so can we use this information to be able to help our supervised approaches within this predicting text difficulty problem. Further analysis is needed to see if we can separate the sentences which are multi-language within the training set to see if we can build better classifiers which have less noise in them and therefore are able to predict the text difficulty with a higher accuracy.

Excluding English stop words within PCA shows the two components smooshed even further on top of each other as shown in PCA **Figure 14** below. This suggests by removing stop words there are overall fewer features and possibly a more clear pattern within the visualization. Both visualizations contain somewhat of a “V” pattern; it is not clear if this is an output of TF-IDF or from some other pattern in our kaggle data set. The “V” pattern is more pronounced within PCA **Figure 14** as it contains fewer features and less “noise” from the stop words.



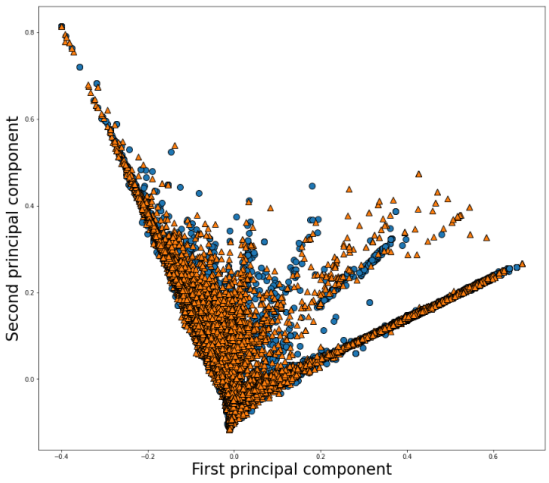


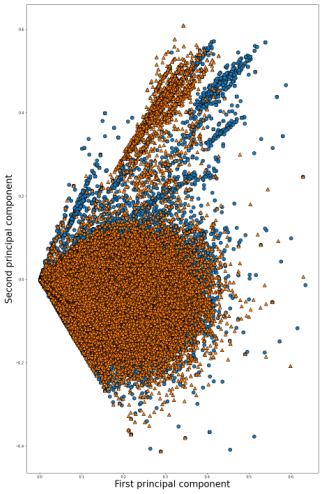
Fig 13 : PCA, n=2, includes stop words

Fig 14 : PCA, n=2, excludes stop words

blue color = difficult / 1 label, orange color = simple / 0 label

Truncated Singular Value Decomposition (Truncated SVD):

We also performed truncated SVD on the TF-IDF vectorizer training data set to look for visual patterns. Unfortunately, this resulted in similar patterns found above within PCA. The graphical results look very similar however they appear to be mostly rotated 180 degrees compared to the PCA plots above.



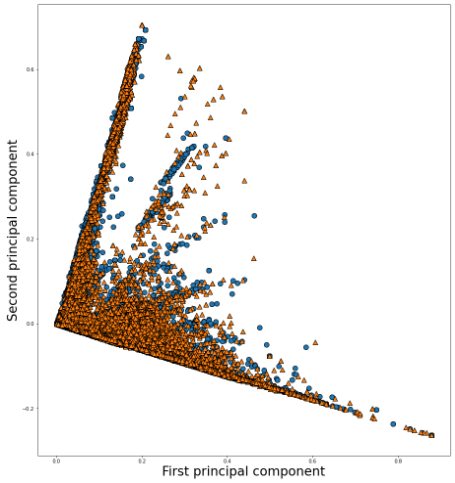


Fig 15 : PCA, n=2, includes stop words

Fig 16 : PCA, n=2, excludes stop words

blue color = difficult / 1 label, orange color = simple / 0 label

LDA: Latent Dirichlet Allocation

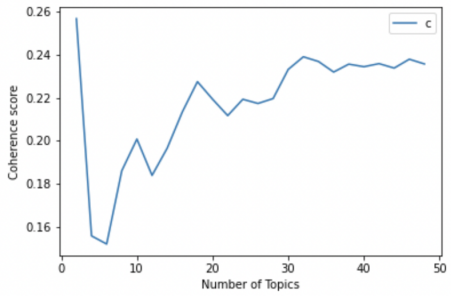
Background:

Topic modeling is a type of statistical modeling to discover abstract topics from the collection of documents. LDA is a popular algorithm for topic modeling and is used to classify text in a document to a particular topic. It builds a topic per document model and words per topic model, modeled as Dirichlet distributions. The challenge, however, is how to extract good quality topics that are clear, segregated, and meaningful. This depends heavily on the quality of text processing and strategy of finding the optimal number of topics.

Model Development and Analysis:

As part of our unsupervised learning analysis, we tried a Gensim based LDA model. The key highlights from this modeling exercise are:

1. Data Preprocessing: convert sentences to words, create bigrams, remove stop words and lemmatize words.
2. Training model using multicore model and compute perplexity and coherence value.
3. Identifying best K topics using Coherence score across 40 topics.
4. Visualize topic allocation between simple and complex sentences.
5. Visualize topics and terms using pyLDAvis package.
6. Derive dominant topic for each document in the training text dataset
7. Compare logistic regression model score, precision and recall graph without and with Topic feature.



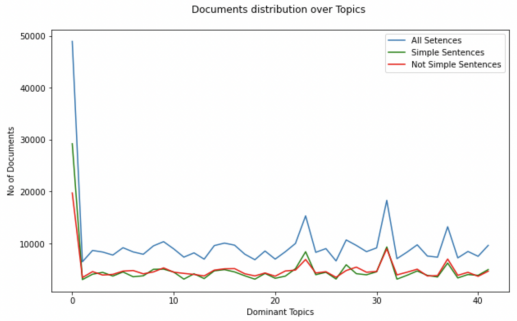


Fig 17 : Coherence Score vs Number of Topics

Fig 18 : Topics distribution for documents type

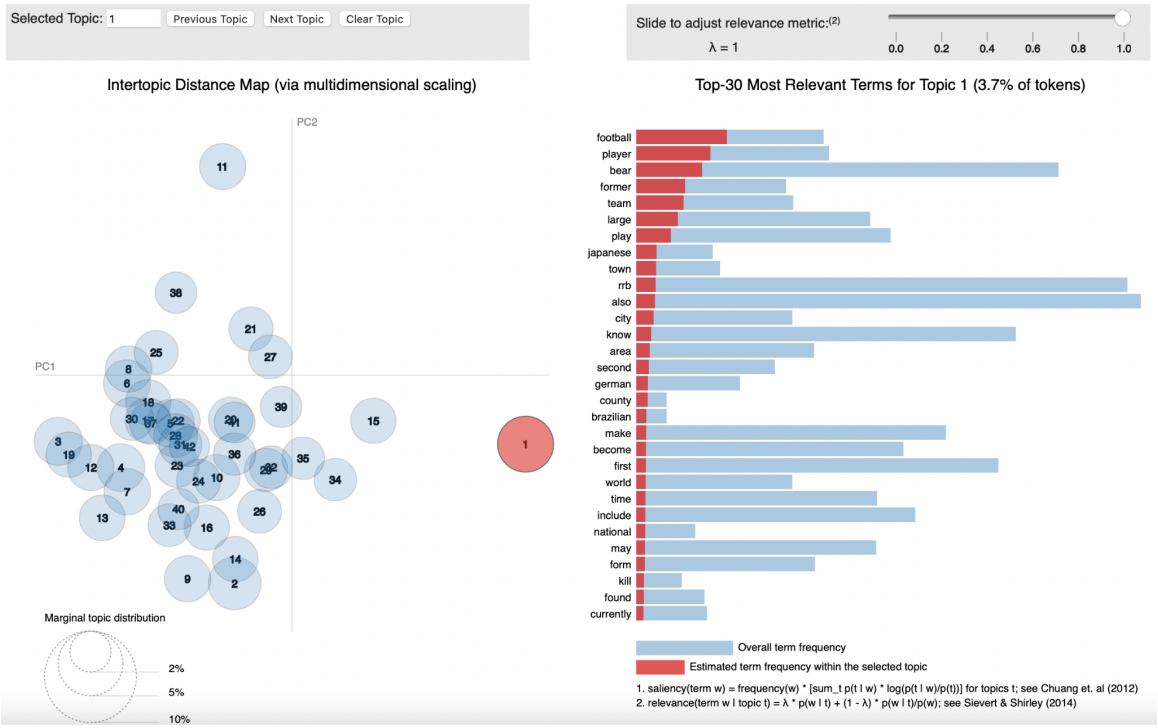


Fig 19 : Top 40 Topics and associated Terms Visualization using pyLDAvis

Insights:

Based on manual inspection and observation, we found the below top 10 Topics across sentences and the top 15 associated words.

Top 10 Topics	Human Identified Possible Topic	Tope 15 Terms
1	Sports	football, player,bear, former, team,large,play,japanese,town,rrb,also,city,known,area,second
2	Player	also,become,release,known,game,year,make,death,play,name,season,music,early,race,first
3	Society	rrb,state,common,use,make,form,also,known,call,often,however,year,family,work,church
4	Interest	make,record,rrb,call,also,include,first,serve,system,known,use,lead,release,may,live,album
5	People migration	time, large,population,make,also,people,move,rrb,city,major,include,period,capital,name,work
6	Government adminstration	use,language,also,official,rrb,japanese,city,call,part,province,main,first,municipality,government,many
7	Social	year, time, first, use,rrb,work,make,name,many,also,become,see,order,new,may
8	Residential	use,rrb,also,know,make,name,large,become,area,may,term,first,feature,locate,place
9	Education	first,know,also,become,receive,may,work,name,season,school,make,state,write,college,call,year
10	Life	state,year,film,also,people,make,include,old,follow,know,become,go,large,rrb,use,many,find,life

Table 2 : Top 10 Topics from LDA

Challenges:

Initially we started with modeling which was not multi core based and it took around 3 hours then we discovered a multicore modeling option which helped to reduce the training time to about 1 hour. After training the model, to expedite the pipeline we save the trained model and load it for subsequent runs to get the best K topics for modeling.

We observed that the LDA coherence score for each execution was not consistent and finding optimum K value was challenging though K=20 seems to be close to be consistent across different executions and hence we chose the same for maximum coherence value. Also having a great lakes platform from the university helped a lot to expedite the model training and evaluation (specially GPU based notebook instance).

Unsupervised Evaluation

LDA: Latent Dirichlet Allocation

Insights: Analysis result suggests that adding sentence topic as a feature from LDA topic modeling did not help to improve Logistic regression model score, precision and recall value.

We thought to test on the simple linear model first to check any impact though we could not try this on Random Forest (selected classifier) due to large retraining time and lack of time. Ideally below two graphs in one plot would have been better to see the difference.

```
Model Score on Test Split: 0.6976439947670999
Precision:  0.7051047637873098
Recall:  0.6828013590467531
Precision Recall Curve: Without LDA feature
```

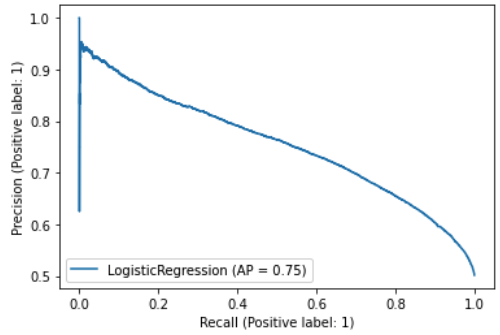


Fig 20 : Precision - Recall without LDA topic

```
Model Score on Test Split: 0.6942714146833255
Precision:  0.7027604541953437
Recall:  0.6767478585442886
Precision Recall Curve: With LDA feature
```

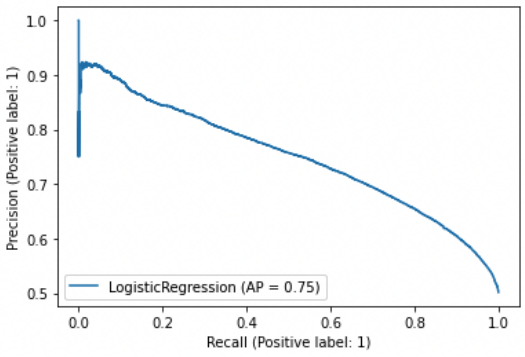
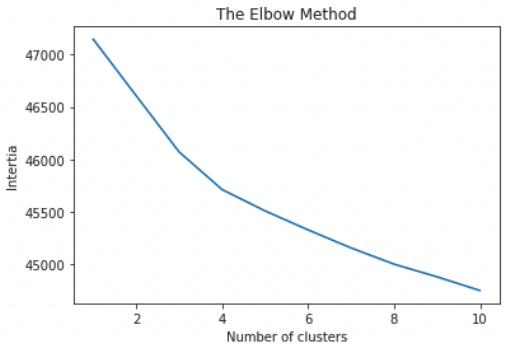


Fig 21 : Precision - Recall with LDA Topics

Clustering

As part of this analysis, we tried to check if the clusters exist in the text dataset. The text dataset is a cleaned text meaning stopwords, punctuations etc. were already removed from it. We used a sparse dataset generated from TFIDF vectorizer on a random sample of 50K training dataset with about 2K features. We used “The Elbow Method” to determine the value of K for K-means clusters.



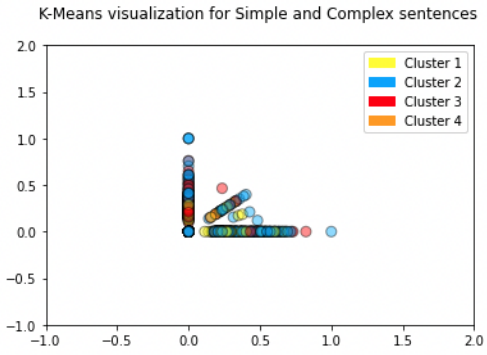


Fig 22 : The Elbow Method

Fig 23 : K-means clustering for 4 clusters

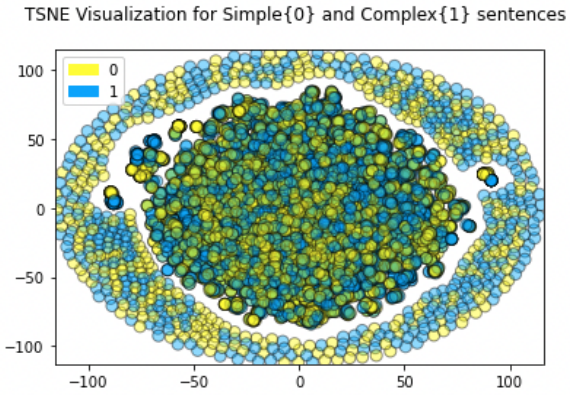
As shown in the figure above, the elbow is less distinct, but 4 appears to be a reasonable number.

Insights:

K-means visualization does not show any clear clusters between simple and complex sentences. Our understanding is that the features derived from the documents are not very similar and hence not able to form distinct clusters, we would like to further investigate to get more insights.

Dimensionality Reduction Visualization

One of the most important steps in high-dimensional data analysis is the exploration of the dimensionality-reduced dataset obtained after tSNE, UMAP or other dimensionality-reduction algorithms. We tried TSNE, UMAP, MDS on a random sample of 10K training dataset vectorized form with 380 features (dimensions).



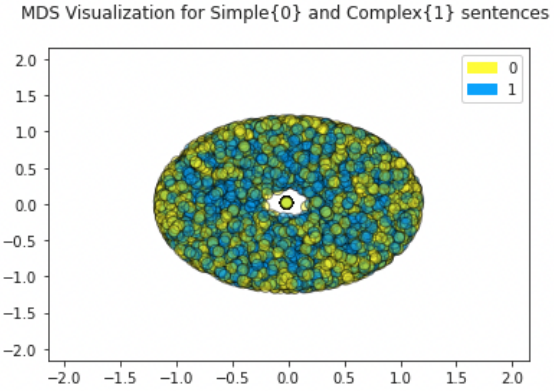


Fig 24 : T-SNE Visualization

Fig 25 : MDS visualization

Insights:

The T-SNE visualization does not find any informative cluster in the features in lower dimension though the empty region between outer ring and center cluster needs to be further investigated. We observed that features from simple and not simple sentences are spread across all the regions. Similar to T-SNE, MDS visualization does not provide any informative features cluster.

From all of the above visualizations, We did not find any clear clusters for feature extraction. We used the default hyper parameters value for these visualizations model and we expect that hyper parameter tuning would give us more insights on the features clustering.

Discussion

What did you learn from doing Part A? What surprised you about your results? How could you extend your solution with more time/resources

Learnings: The key learnings from supervised analysis techniques is that feature engineering is a critical step during model development and we observed that by including relevant features the model performance improved. We combined the features extracted through unsupervised learning into supervised learning methods (e.g Topic feature from LDA model) which also gave us good exposure to the methods of combining both learnings from both the models.

Surprises: Stopwords removal did not improve the model performance hence we retained stopwords to train the classifier. We thought that topics as features from unsupervised learning would improve the score but to our surprise the score did not improve. The classifier selected did not show significant changes with parameters max_depth and we allowed the classifier to choose the depth.

Future Improvement: Given more time and resources, we would like to perform more hyper tuning of the visualization models to derive better insights. We may perform significant tests including confidence/standard error for each model to at least estimate the difference in accuracy across models that are statistically significant. We would also aim to train other classifiers like Logistic Regression and MLP to compare the Precision-Recall curve . This may provide us better insights to the classifiers and applicability to the given problem.

What did you learn from doing Part B? What surprised you about your results? How could you extend your solution with more time/resources?

Learnings: The key learnings from unsupervised analysis techniques we tried (PCA, LSA, LDA and Visualization) is that vectorized text features reduction from high dimensionality to lower dimensions to obtain terms capturing the maximum variation in the data is not quite obvious.

Surprises: None of the dimensionality reduction visualization provided a clear clustering on the vectorized text features.

Future Improvement: Given more time and resources, we would like to perform more hyper tuning of the visualization models to derive better insights and leverage the same for extracting features for supervised learning. We would like to explore more LDA based features which can be leveraged for supervised learning algorithms and compare them against different models for different metrics.

Ethical Considerations

Part A and B: Supervised Learning and Unsupervised Learning

Our assumption is that the current percentage of false negative and false positive is not an optimal solution to the problem and needs further improvement. The selected classifier outcome would still require human interventions to review the predictions made by the classifier and label accurately. The model output would still be helpful in reducing the efforts for classifying unlabeled texts into simple and

not simple but reviewing simple sentences before handing over to the kids is recommended based on the current accuracy of the model.

For eg : The impact with the current classifier would be huge if the classifier is implemented to simplify kids' education material and a complex text is labeled as simple which probably would result in reduction in kids interest in the educational material (e.g. text books). That calls for the reviewer's involvement to review the simple sentences classified by the model.

Statement of Work

Pankaj Kumar Singh	Pierre LeBlanc	Biswajit Kumar
Feature Engineering: Countvectorizer, TFIDF, Extracting features from AoA and Concreteness dataset. **Supervised Modeling**: Basic model checking for Logistic Regression, Dummy classifier, Random Forest, Doc2Vec, Neural Network, Feature Importance **Unsupervised Modeling**: LDA topic modeling, Visualization: TSNE, MDS and Kmeans **Failure Analysis**: LIME based analysis for Logistic and Random Forest predictions	**Feature engineering:** TF-IDF, word count 3rd party data (dale_chall) Complexity of grammar **Supervised methods:** Pretrained BERT Random Forest Doc2Vec MLP (Neural Network) Logistic Regression Dummy **Unsupervised methods:** Visualization and analysis of PCA and Truncated SVD **Failure Analysis**: LIME based analysis for BERT pretrained model	**Feature Engineering**: TFIDF, Extracting features from AoA and Concreteness dataset, syllables as feature **Supervised Modeling**: Compare models Logistic Regression, Dummy classifier, Random Forest, Neural Network, Decision Tree NaiveBayes,GradientBoosting, Hypertuning the parameters Model performance and Evaluation Sensitivity analysis Learning curve **Unsupervised Modeling**: Visualization: Basic TSNE, MDS and Kmeans (from SIADS 543) **Failure Analysis**: Confusion Matrix for trained classifier Code review and consolidation of supervised methods

References

1. https://www.kaggle.com/competitions/umich-siads-696-f22-predicting-text-difficulty/overview
2. Introduction to Machine Learning with Python, O'reilly, September 2016: https://learning.oreilly.com/library/view/introduction-to-machine/9781449369880/
3. BERT explanation from Chris McCormick: https://www.youtube.com/watch?v=FKlPCK1uFrc

   Source code of BERT from Chris McCormick: https://mccormickml.com/2019/07/22/BERT-fine-tuning/

4. Great Lakes: *This research was supported in part through computational resources and services provided by Advanced Research Computing at the University of Michigan, Ann Arbor.*

5. UMICH MADS SIADS 505,501,522,532,543,542 for Python Programming

6. LIME: https://marcotcr.github.io/lime/tutorials/Lime%20-%20basic%20usage%2C%20two%20class%20case.html

7. Link for all source code: https://drive.google.com/drive/folders/1K6-a3xTWoJc_Y6dJdqk4m81EIy43PRxS?usp=sharing

Appendix A

1. Data source complete descriptions from Kaggle competition web site (source link also shown below):

https://www.kaggle.com/competitions/umich-siads-696-f22-predicting-text-difficulty

2. Features

a. Twenty one features Extracted:

original_text_cleaned,sentence_length,dale_words_prcnt,aoa_words_prcnt,aoa_words_age_scaled,aoa_words_freq_scaled,aoa_words_syll_scaled,con_words_prcnt,con_known_prcnt_scaled,Conc,con_mean_count_scaled,con_subtlex_count_scaled,diff_unknown_words,simp_unknown_words,avg_syllable,syllable_1,syllable_2,syllable_3,syllable_more_than_3,,noun_words_prcnt_scaled

b. Final Thirteen features for models after feature ablations

original_text_cleaneds,sentence_length,dale_words_prcnt,aoa_words_prcnt,aoa_words_age_scaled,aoa_words_freq_scaled,aoa_words_syll_scaled,con_words_prcnt,con_known_prcnt_scaled,con_mean_count_scaled,con_subtlex_count_scaled,avg_syllable,noun_words_prcnt_scaled

3. BERT layers within Bert base (example shown below):

```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.01, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.01, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.01, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.01, inplace=False)
          )
        )
```

```
        [layers repeat 11 more times here…for a total of 12 layers]
                    ….
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.01, inplace=False)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
)
```

4. Dataset Description:

Training data set: "WikiLarge_Train.csv" contains 416,768 English (mostly English but does contain other languages as well) sentences with a label for each sentence:

0: the sentence does NOT need to be simplified.

1: the sentence DOES need to be simplified.

The test data contains "WikiLarge_Test.csv" 119,092 sentences that are unlabeled.

3rd party data sets: provided by the Kaggle competition web site: (more info on these is in Appendix A)

dale_chall.txt : *"This is the Dale-Chall list of ~3000 elementary English words that are typically familiar to 80% of American 4th grade students (in the 90s)" (source Kaggle)*

Concreteness_ratings_Brysbaert_et_al_BRM.txt *: "Concreteness ratings for about 40k English words." (source Kaggle)*

AoA_51715_words.csv *:: "List of approximate age (In years) when a word was learned, for 50k English words." (source Kaggle)*