Text Difficulty Analysis Through Supervised and Unsupervised Learning

By: Greg Knox, Aaron Evans, and Nicolas Calo

University of Michigan School of Information
Master of Applied Data Science

SIADS 696
October 24, 2022

Table of Contents

Introduction

For our project, we chose to examine text complexity using Wikipedia data. We hoped to build a classifier that could label a sentence as being in simple English or not in simple English. This will save time for a human editor, who would eventually like to place the simplified article in the simple Wikipedia, which is used for those with learning disabilities or those who use English as a second language. We used various methods of analyzing the sentences taken from Wikipedia including logistic regression, random forests, and deep learning models to examine what sentences were considered complex and which sentences could be considered simple. Unsupervised methods, such as truncated singular value decomposition (SVD) and Latent Dirichlet Allocation (LDA) were used to reduce dimensionality and focus on subsets of the data. Once reduced, K-means clustering examined the natural clusters in the data to see if those clusters correlated with the complexity categories in the data.

Data Sources:

For this project, we were provided with multiple data sources from the project's Kaggle homepage: https://www.kaggle.com/competitions/umich-siads-696-f22-predicting-text-difficulty

They included:

- WikiLarge_Train.csv- This was our sample training set. It included 416,768 sentences, which were labeled 0, which meant the sentence did not need to be simplified, or 1, which meant that the sentence needed to be simplified. Our fields for the training and test sets also included an id field, as a unique identifier, a text field, which is the sentence in free-form text, and a label class, which was either 1 or 0.
- WikiLarge_Test.csv- This was our test training set, which was not labeled.
- sampleSubmission.csv- This was simply a sample submission file in the correct format for submission to Kaggle, which scored our test data.
- dale_chall.txt- A list of 3,000 English words that are considered familiar to eighty percent of fifth grade students.
- Concreteness_ratings_Brysbaert_et_al_BRM.txt- concreteness ratings for forty thousand English words
- AoA_51715_words.csv- Age of Acquisition" (AoA) estimates for about fifty-one thousand English words. File contains approximate age (in years) when a word was learned, with the earlier considered to be more basic and hence, a lower score.
- HLT 2004 Readability: Unigram Language Models- A file with frequency counts for the twelve categories of labeled documents for grade-level language models.

We note that in the analysis below, we focused on the training data and the Dale-Chall data.

Exploratory Data Analysis

Our initial focus was to examine the raw data. Using pandas, we read in the csv file and noted the shape, which was 416768 rows and 2 columns. The labels were ‘original_text’ and ‘label’, plus an index for each instance.

Upon initial examination of the data in the pandas DataFrame, we noted that many of the sentences were not full sentences. Many just contained punctuation. Rather than an extensive text pre-processing phase to begin, we suspected a TfidfVectorizer would help for our initial model run. We used English stopwords, a unigram/bigram model, and 20,000 features as our initial hyperparameters. Later, we vectorized the Dale-Chall list to our features, as well. To do this, we initially cut down our features to 5,000 and went back to a unigram model. Once we ran our first round of unsupervised models, we also cleaned the text a bit more by putting the text in lowercase and stripped out the ‘lrb’ and ‘rrb’ terms, which were initially prominent feature names in our clusters. However, ‘lrb’ and ‘rrb’ appeared to be simply bracketing mechanisms in Wikipedia’s markup language. We will discuss more on this below.

Part A: Supervised Learning

We started off with just taking the data and running it through a TF-IDF vectorizer and doing basic analysis to set a baseline of understanding. When we tried multiple dummy classifiers, we ended up with around a 50% success rate.

Motivation

For Part A we wanted to take a look at the different ways to evaluate the sentences and see how we could best learn what sentences needed to be simplified and which ones did not. We wanted to see what methods and what features lead to the best scoring of data as it related to complexity. To start, we just took a simple look at a vectorized set of training data and ran it through Dummy Classifiers to get a baseline evaluation. We then moved on to Logistic Regression to get a simple and quick model done. From there, we also tried more complicated models, such as Random Forests and Deep Learning.

Data Source

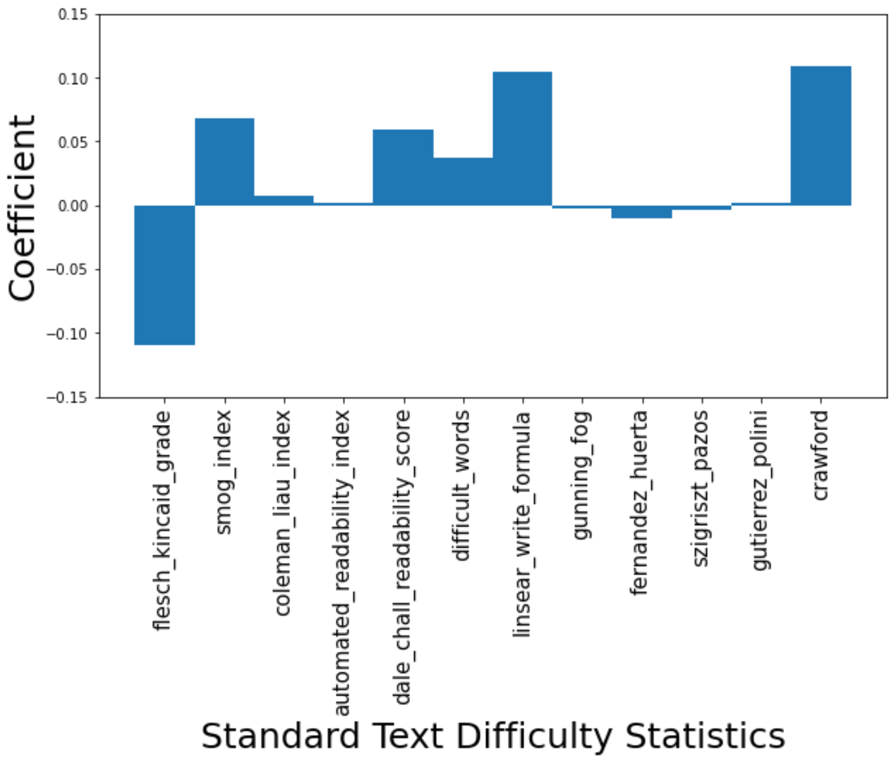
For Part A, we used the WikiLarge_Train.csv, WikiLarge_Test.csv, and dale_chall.txt files provided in the Kaggle project.

Methods & Evaluation

Initial Run: Logistic Regression

In order to get an initial accuracy score for a baseline model, we set up a logistic regression model. We set the hyperparameters in the Tfidfvectorizer to have English stopwords, to use both unigrams and bigrams, and have 20,000 features. We split our training data into training and validation sets, with 80% of examples in the training set and 20% in the validation set. For the logistic regression classifier, we set the maximum number of iterations for the solver to 1,000. Our training set scored in the roughly 72% range and our validation set scored approximately 70%. When we vectorized and transformed our testing set, we ran it through Kaggle and scored 70.01%. Although we considered this a fairly good initial score, we did not anticipate this being our highest scoring analysis overall.

Textstat text statistics



Textstat is a library used to calculate traditional text difficulty statistics. Texstat was utilized to extract the statistics listed on the x axis ticks of the image on the right. The image is a bar chart displaying the logistic regression extracted coefficient of each feature. The features with low coefficients values were removed and the logistic regression model was rerun. As features with low coefficients have lower effects on the dependent variable. The model continued to perform relatively the same. Overall, both the model with all textstat features and the filtered model returned around 65% accuracy.

Dale-Chall:

In another attempt to improve our logistic regression accuracy, we imported and vectorized the Dale-Chall word list. We did the vectorization by starting from our TF-IDF vectors and replacing the non-zero values with a +1, if the word was in the Dale-Chall list, and a -1 if it wasn't in the

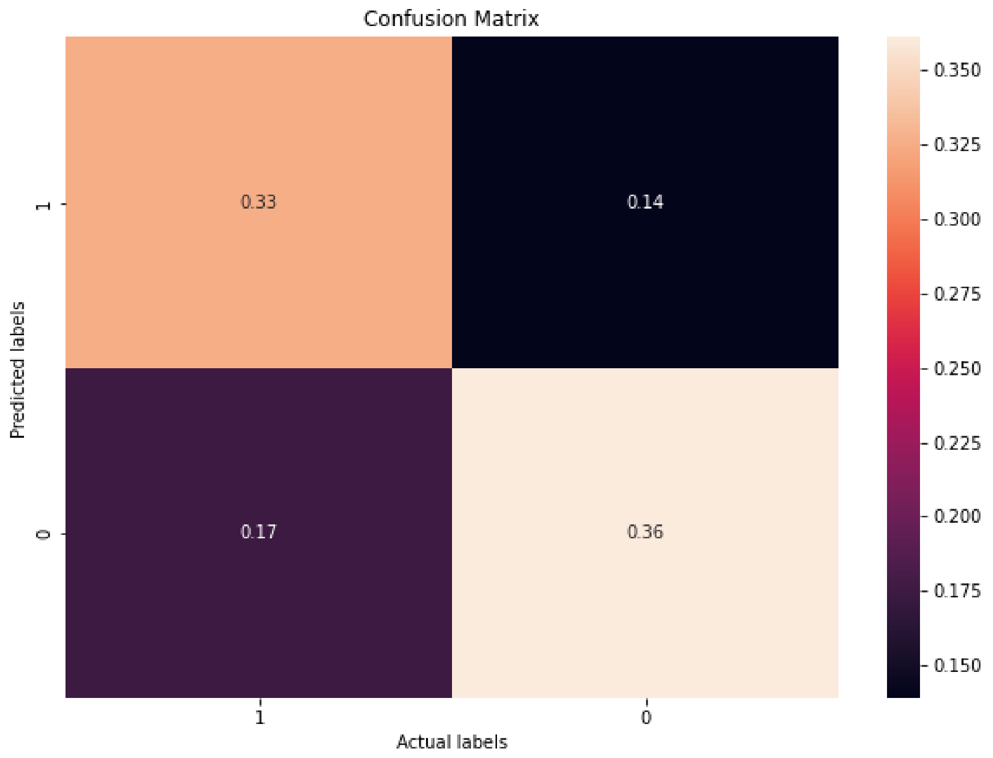
Dale-Chall list. We had high hopes for the Dale-Chall list, since this is a list of 3,000 words familiar to 80% of fifth grade students. We believed this would help our logistic regression classifier better classify our sentences as simple or not simple. However, since this doubled our number of features, we initially cut our initial vector to 10,000 features and only used unigrams, as it was initially unclear as to how to apply our Dale-Chall scoring scheme to bigrams. Our scores were slightly worse than the baseline, or 71.01% for the training set and 67.85% for the validation set. Later, we tried some further text pre-processing. This included replacing hyphens with spaces, removing other non-word characters and lemmatizing each word. We hoped to go beyond the simple clean-up afforded by the TfidfVectorizer, but our scores dropped again, so we abandoned the more advanced text pre-processing and stuck with the TfidfVectorizer. Later, we would use a bigram version of the Dale-Chall, in which we assigned a value of +2 to bigrams, where both words were in the Dale-Chall list, a value of +1 where neither word was in the Dale-Chall list, and a +1.5, if one was in the list and the other was not in the list. However, that version still did not perform as well as our first model run, scoring 69.66% in our training set and 67.73% in our validation set. However, none of these initial analyses were cross-validated, so they may not replicate when studied more rigorously.

Feature Importance

	No Dale-Chall		Dale-Chall	
	Train	Val	Train	Val
	Mean (SD)	Mean (SD)	Mean (SD)	Mean (SD)
5k Base Features, Unigrams	.6869 (.00072)	.6722 (.0012)	.6927 (8.7 e-5)	.6726 (.0016)
5k Base Features, Bigrams	.6909 (.00076)	.6764 (.0017)	.6976 (.00056)	.6782 (.0012)
10k Base Features, Unigrams	.7020 (.00040)	.6764 (.0014)	.7103 (.00029)	.6787 (.00083)
10k Base Features, Bigrams	.7081 (.00043)	.6832 (.00085)	.7172 (.00059)	.6853 (.0018)
20k Base Features, Unigrams	.7143 (.00030)	.6746 (.0017)	.7288 (.00052)	.6766 (.0014)
20k Base Features, Bigrams	.7245 (.00037)	.6845 (.0025)	.7388 (.00047)	.6870 (.00092)

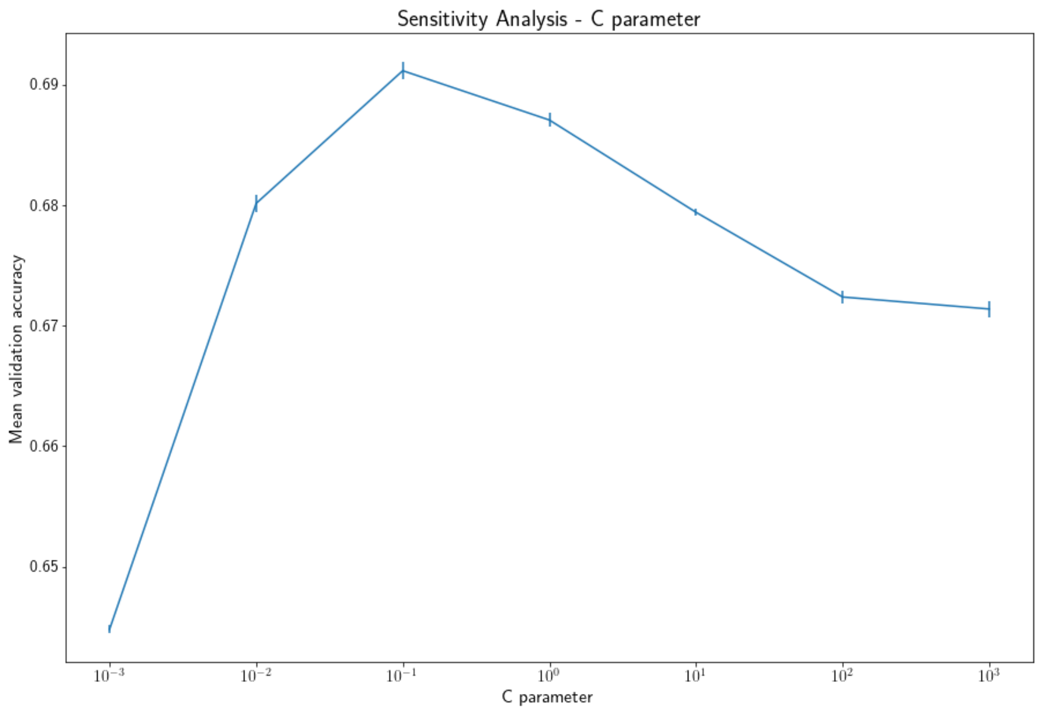
After initial exploration of the data set, we decided to be a bit more formal. We added in five-fold cross-validation and systematically explored the influences of unigrams versus bigrams, base features (number of features produced by the Tfidfvectorizer), and whether we should double the feature size by adding in the Dale-Chall features. Results are shown in the table above (averaged over the five-folds for each analysis). Generally, there was a positive effect on validation performance for increasing numbers of base features and for bigrams versus unigrams, and including the Dale-Chall features over the No Dale-Chall features. However, these benefits were modest, with only a difference of ~1.5% between the worst and best models.

Failure/Error Analysis



To assess whether there was anything systematic about the errors made by the logistic regression model, we took the best performing model from the feature importance analysis and generated a confusion matrix for it. In general, there did not appear to be strong bias in terms of how often the model predicted sentences that needed to be simplified versus those that did not need to be simplified. Overall, the rate of predicting that sentences needed to be simplified was slightly lower, at 47%, versus those that did not need to be simplified, which was 53%. Correspondingly, there were slightly more false negatives (17%) than false positives (14%).

Sensitivity Analysis



To determine how sensitive our analyses were to variations in hyperparameters, we again ran our best performing logistic regression model, with five-fold cross-validation at seven different values of the C regularization parameter, from .001 to 1000 in multiples of ten. The best performance was at C = .1, although the second-best performance was at C = 1, which is the default value in Scikit-learn and the value used in the analyses above. Accuracy dropped off sharply at C = .001, but otherwise the effect of regularization was moderate. The difference between C = 1 and C = .1 was only .4%, and except for C = .001, the entire range was within 2 percentage points of accuracy. However, for C values greater than 1, the optimizer warned us that it reached the maximum number of iterations without converging, so higher accuracy values may be possible at those values of C, if the maximum number of iterations were increased. We also intended to explore the effect of L1 versus L2 regularization, but the ‘saga’ solver needed for elastic net regularization failed to converge in any reasonable amount of time. We were able to complete an analysis of fully-L1 regularization versus fully-L2 regularization with the ‘liblinear’ solver, which does not permit a mix of L1 and L2 regularization. The L1 regularization performed marginally better than the L2 regularization with validation accuracy of 68.9%, versus 68.6% for L2 regularization, which is the default and the regularization parameter used in the analyses above.

Deep Learning:

Multi-Layer Perceptron Model

We believed that deep learning models might hold a key to higher scores. Using TensorFlow and Keras, we initially designed a multi-layer perceptron neural network model. Our sequential model was designed to have two hidden layers of 32 and 16 units with an input shape of 10,000, of which 5,000 were the original Tfidf vectors and 5,000 Dale-Chall vectors. Our activation functions for these hidden layers were both ‘relu’. Our output layer was 2 units and we used the ‘softmax’ activation function. Our loss function was categorical cross-entropy with an ‘Adam’ optimizer. We trained our model using a batch size of 200 and a maximum of 10,000 epochs. However, we also used early stopping, and the ‘reduce learning rate on plateau’ feature, with patience for the early stopping set at 30 epochs and the patience for the learning rate reduction set to 10 epochs. The learning rate reduction factor was .5. For both early stopping and learning rate reduction, the patience value applied when the loss value on the validation set had not improved for that number of epochs. Thus, more realistically, the epoch count was in the 30’s because the performance on the validation set plateaued rather quickly. While this took a few minutes to run, we noticed that our training set accuracy continued to increase, topping out at 94.42%, while our validation accuracy sputtered at 65.47%. We were clearly overfitting our model. To attempt another model, that may not be as prone to overfitting, we next tried a convolutional neural network.

Convolutional Neural Network Model

In our initial attempt at a convolutional neural network, in order not to exceed our machine’s memory capacity, we reduced our number of samples down to 20,000 in the training set. We cut down our features to 5,000 Tfidf features and 5,000 Dale-Chall features, in order to conserve memory. Our first hidden layer was a convolutional layer with 3 units, a filter size of 100 x 2 and a stride of 50 x 1. To fit our convolutional architecture, we re-shaped our input data to 5000 x 2, with the Tfidf features lined up with their corresponding Dale-Chall features. Our hope was that the convolutional network might be able to form some sort of integrated representation between the Tfidf features and the Dale-Chall features based on their spatial correspondence. The output from the convolutional layer was flattened and used as input into another dense hidden layer with 16 units. Our hidden layers used a ‘relu’ activation function and our output layer used the ‘softmax’ activation function. We again used an ‘Adam’ optimizer and a loss function of ‘categorical cross-entropy’. The early stopping and learning rate reduction were the same as our multi-layer perceptron model. While this model did not overfit as much, as our training set accuracy score topped out at 70.11%, our validation accuracy was only 63.45%. We ran this model through the Great Lakes Cluster, hoping with greater computing power, we could use more features and train on all of the data. This allowed us to go up to 50,000 training samples and 20,000 features (10,000 Tfidf and 10,000 Dale-Chall). While this helped to a degree, our training accuracy improved to 80.08%, while our validation accuracy only improved to 64.59%. However, some of us on the team had never used a computing cluster before, so this was a worthwhile exercise.

Random Forest Classifier Model

We also used the vectorized test data to train several Random Forest algorithms. We chose to use Random Forest for its ability to handle data with many variables. Random Forests can handle missing variables over the dataset well. The main difference between the different forests was the max depth of the trees in each run. There was also the difference between using the Dale-Chall data and just testing the test data on its own. Initially, we tried to look at trees that had a max depth of two and unsurprisingly, the results of those efforts were not very good, having only around 60% success for both sides of the train-test split. A max depth of 25 only improved the results slightly. Increasing the max depth to 200 was what greatly improved the results for the train scores. We were able to get up to 85% for the train data but the test data results were still only around 65%. This showed us that the model was overfitting the training data and was not able to adapt to the new test data very well.

Here are the confusion matrices for the 200 depth trees with and without the Dale-Chall data:

Without Dale-Chall	Actual 0	Actual 1
Predict 0	30833	10852
Predict 1	17016	2453

With Dale-Chall	Actual 0	Actual 1
Predict 0	31109	10616
Predict 1	17422	24207

The confusion matrices show that with the Dale-Chall data, the 200 depth Random Forests are more likely to predict a sentence will need to be simplified when it does not.

Part B: Unsupervised Learning

Motivation

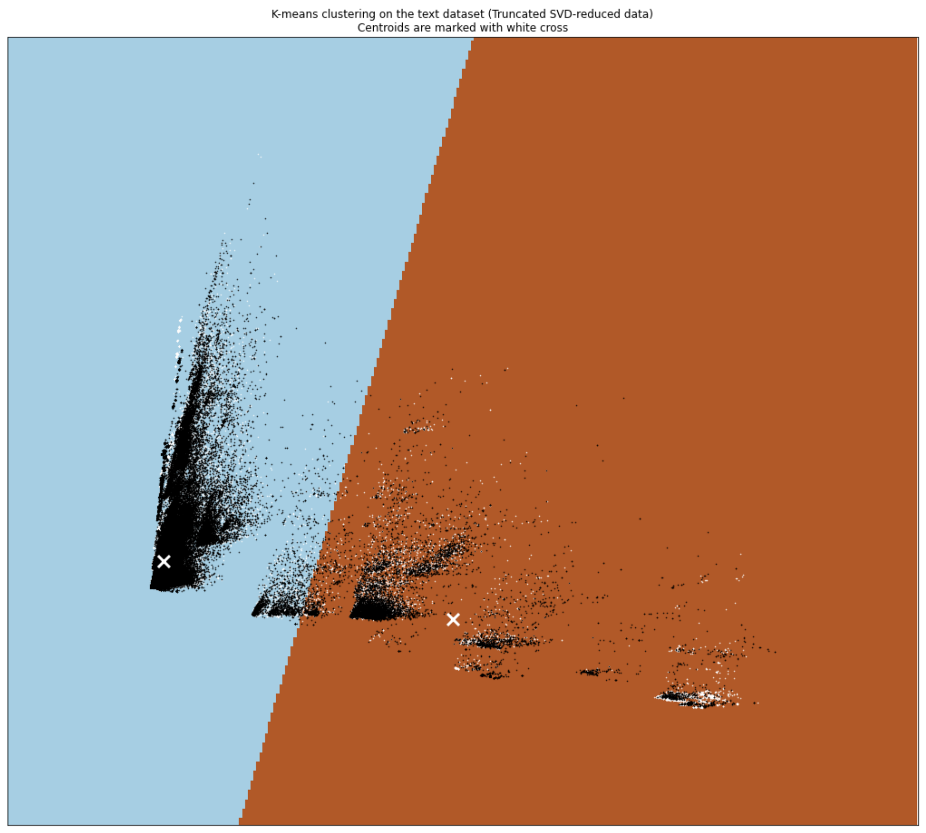
Our goal for utilizing supervised learning was two-fold. First, we wanted to be able to visualize our data in a vector space. We wanted to be able to see how similar the non-difficult and difficult sentences generally were. Second, we wanted to utilize topic modeling to extract topics from sentences to implement as features in a supervised learning model.

Data Source

For Part B, we used the WikiLarge_Train.csv and WikiLarge_Test.csv files provided in the Kaggle project.

Unsupervised Learning Methods:

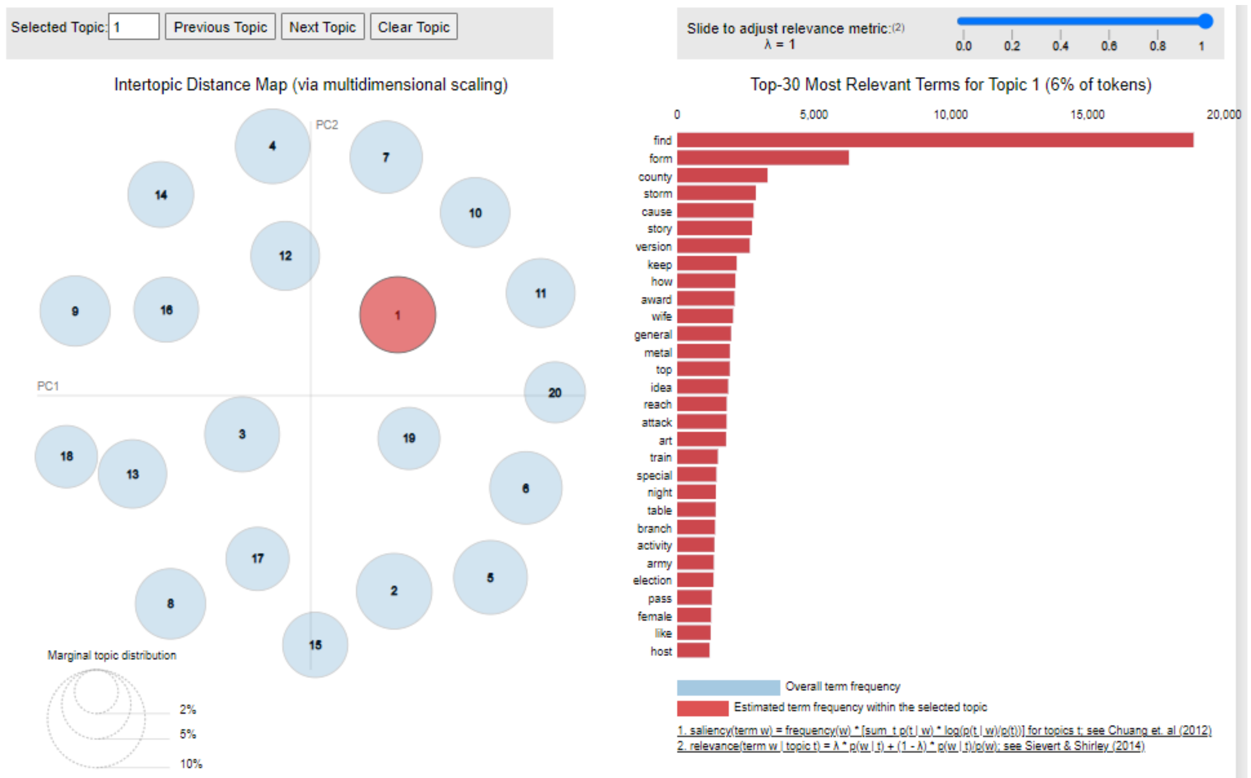
Truncated SVD with K-Means Clustering



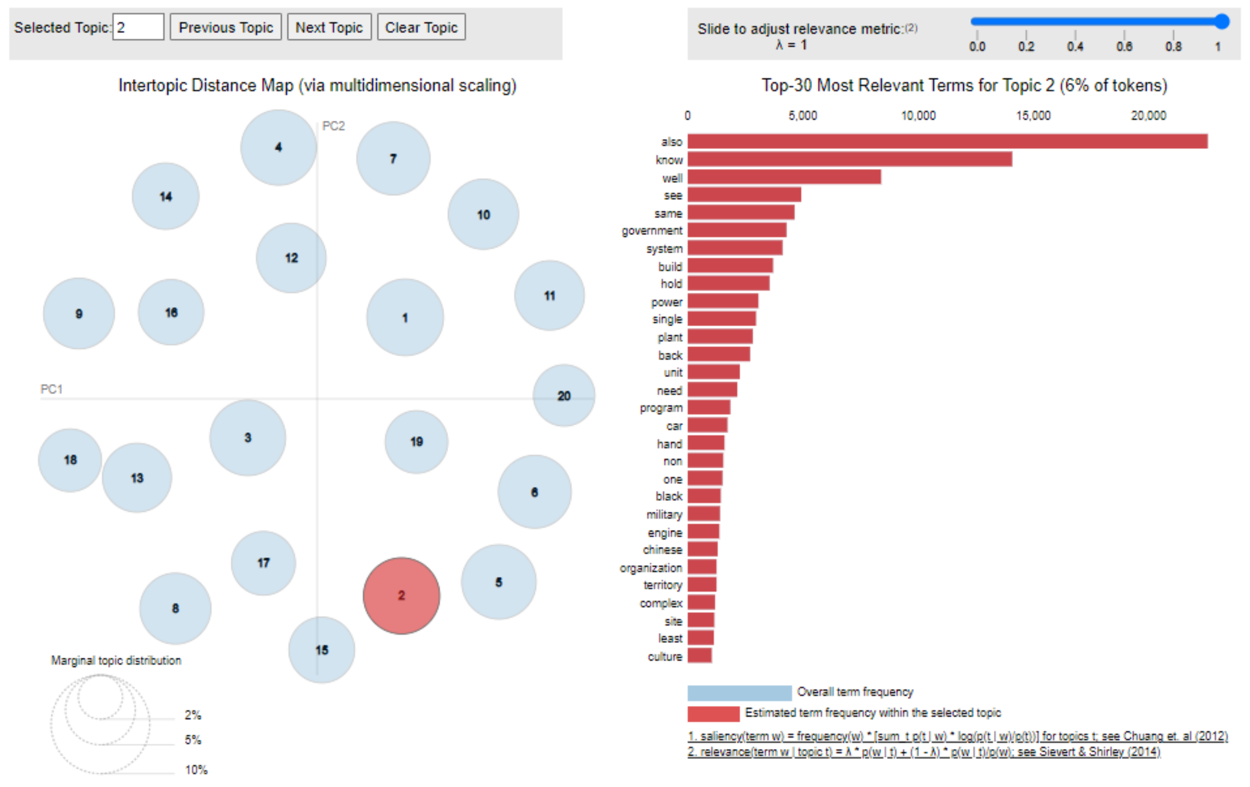
Because the initial number of features was large, we decided to cut down on those features through a principal component analysis. However, due to the sparse nature of our matrix, we used Scikit-learn's truncated singular value decomposition (SVD), instead. Truncated SVD with two dimensions as a hyperparameter allowed us to reduce dimensionality and focus on subsets of our data. Once our dimensionality was reduced, we ran our reduced data through a K-means clustering model to see if our important features fit neatly into our two-cluster representation. They did not. Unfortunately, while there seemed to be two, neatly formed clusters, plus a less-formed third cluster, the results were not what we hoped for. It did not appear that the dimensions of the K-means analysis were related to the complexity of the sentences. Both complexity classes were intermingled on both sides of the decision boundary, rendering our analysis rather worthless. We also tried running the Truncated SVD with three dimensions to see if that improved our results, but that also did not work well. When we sorted our clusters for feature names in each cluster, we realized 'lrb' and 'rrb' appeared in the results. When we researched these terms, we realized they were the bracketing mechanism in Wikipedia's markup language and cleansed the data of these terms and re-ran the models. This gave us cleaner and different feature names, but was still unsuccessful.

LDA

In order to try to improve upon our accuracy we implemented Latent Dirichlet Allocation (LDA). The goal was to try to identify the abstract topics within the data set and use the model to predict the probability values of the abstract topics for the test set. Before doing this we wanted to thoroughly clean the text data. Any text item that would not be classified as a noun, adjective, verb, or adverb was stripped. Spacy was then employed to lemmatize words. We then used Gensim.utils.simple_preprocess to convert each stripped sentence into a list of tokens. Bigrams and trigrams were then formed using gensim.models.phrases.Phraser. An LDA model is then trained using gensim.models.ldamodel.LdaModel. The challenge with Gensim was the operations did not allow multithreading. So most operations took a long time. Running code through took about 3 hours, even while using the Great Lakes computing cluster. The model is then utilized to return the probability of topic values and to create the two visualizations below. The two visualizations below display the topics (in this case 20) displayed in a vector space (on the left half of each image) and the most relevant terms for the highlighted topic bubble (on the right half of each image). These visualizations were created with pyLDAvis. These probabilities and the labels are then utilized to train both a logistic regression classifier (sklearn) and a neural network classifier (tensorflow). Unfortunately this only resulted in an accuracy of 60%. We tried variations of hyperparameters and many variations of neural network layers and quantities, but saw very little change in accuracy. Fewer layers with a “relu” activation function and an “adam” optimizer performed the best. In a separate notebook, this was all also run without the bigram and trigram operation. For this separate run TFIDF was also used to remove frequently occurring words. Both notebooks resulted in similar accuracies.



1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)


Selected Topic: 2
Previous Topic
Next Topic
Clear Topic
Slide to adjust relevance metric:(2)
λ = 1
0.0
0.2
0.4
0.6
0.8
1
Intertopic Distance Map (via multidimensional scaling)
PC2
PC1
4
7
14
10
12
11
9
16
1
20
3
19
18
13
6
17
2
5
8
15
Marginal topic distribution
2%
5%
10%
Top-30 Most Relevant Terms for Topic 2 (6% of tokens)
0
5,000
10,000
15,000
20,000
also
know
well
see
same
government
system
build
hold
power
single
plant
back
unit
need
program
car
hand
non
one
black
military
engine
chinese
organization
territory
complex
site
least
culture
Overall term frequency
Estimated term frequency within the selected topic
1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

Unsupervised Evaluation

LDA

Utilizing LDA allowed us to separate our sentences into a set number of topics. We utilized the probability of each sentence being assigned to each topic as features for supervised learning models. The two model types utilized were logistic regression and a neural network.

LDA does not appear to lead to a high level of accuracy when utilized as feature input for a supervised learning model. Changing hyperparameters within the LDA yielded similar results. Changing passes from 8 to 12, increasing chunksize from 80 to 300, and increasing the number of topics from 20 to 50 all had little effect on the overall accuracy. Unfortunately LDA did not lead to improved results. This seems to be due to overlapping of topics within the vector space. As we were unable to separate difficult sentences from non-difficult sentences utilizing LDA.

Discussion

What We Learned from Part A

Part A taught us about the difficulty of understanding what it is that is useful in evaluating text data. It was not an easy task to look at the data that we were given and come up with a quick solution to which sentences were simple and which were complex. We learned about the complexity of the data itself and how sometimes we do not always get properly labeled data so that can throw off our results.

With further time and effort, it would have been good to go through the data that we received and do a better job of cleaning the training section and work on more clear parameters from the start as to what constituted a simple sentence and a complicated sentence. One thing that we tried to do was to implement the Stanza parser to work with the data but it took a long time to figure out how to install and use it that we did not have enough time to to truly learn how to use it. Further work would have tried to actually use the parser to get a better understanding of the sentences that we were working on.

Working through Part A also taught us a lot about working with data and with packages. We learned how to troubleshoot importing packages and using different packages involved with our work.

What We Learned from Part B

In the initial run we were able to visualize the overlap of sentence difficulty labels within a PCA initialized vector space. This visualization makes us believe there may be a decently high proportion of sentences that are mislabeled. This would make sense as the definition of a difficult sentence would be different from person to person.

Through our analysis of the data using LDA, the topics within difficult and non-difficult sentences seem to overlap quite a bit. Unfortunately this means utilizing LDA to identify topics within sentences will not do a great job separating difficult and non-difficult sentences.

Part B taught us to really test and question the validity of the data we receive to work with. We discovered various ways to visualize this uncertainty utilizing unsupervised learning methods. Unfortunately we were not able to greatly increase the accuracy of our supervised learning models utilizing unsupervised learning.

Ethical Considerations

There do not seem to be any major ethical concerns about the data that we used. We received the data from a Kaggle competition but it was originally sourced from Wikipedia, and our use of the data, in this context, is permissible within Wikipedia’s license and guidelines. Ethical concerns may creep into the project when we start talking about the evaluation and models used to study the data. The wrong standards could be used to evaluate the complexity of the data which could lead to misinterpretation and misrepresentation of words and texts once the project was complete. If someone were to deploy a machine learning model utilizing this dataset, they should be cautious to ensure the difficulty of sentences in this dataset reflects the level of difficulty they should be expecting within their application space. In general, people using machine learning should take care that their models are not used in ways that marginalize people, based on specific protected classes. For instance, if this is utilized for a job application filter, be sure this does not unfairly discriminate against a specific group of people.

Statement of Work

Greg Knox performed most of the initial data cleansing, logistic regression models, multi-layer perceptron, and convolutional neural network models, as well as the Truncated SVD and k-means clustering analyses.

Aaron Evans worked on logistic regression models and random forests. He also looked into different ways to look at the features of the sentences and figuring out how to parse them using Stanza.

Nicolas Calo utilized textstat to draw out text difficulty statistics from sentences. Performed various supervised learning experimentation. Utilized LDA to predict abstract topic vector probability values. These values were utilized to train a supervised learning model to predict sentence difficulty. Both logistic regression and a neural network were utilized separately.

All three members contributed to the write-up and editing of the report.