Predicting Text Difficulty with a Focus on Lexical Complexity Feature Engineering

Dylan Kinder and Nicholas Dunbar

1. INTRODUCTION

Rather than focusing on tuning a variety of models to achieve the highest performance, we opted to spend considerable time and effort on feature engineering by exploring an intuitive theory behind word complexity to create a complexity score feature for each sentence. In essence, the linguistic natural pattern of language development concludes that a word's complexity is roughly as complex as the thing it is trying to express. In primitive languages, there is only a need for primitive words which help speakers describe their immediate environment and needs. Our language was forced to mature to meet the need for increasingly more complex systems, and as such we approached the task of classifying complexity primarily at the individual word level under the assumption that a word's complexity is directly proportional to the complexity of the words required to define it. Theoretically, the words used to define new words evolved prior to the word they define, meaning from an evolutionary perspective they should theoretically be less *complex*. This is explored further by psychologist and linguist, Steven Pinker, in his book, "Words and Rules".

Even with this focus on feature engineering, we wanted to ensure that we utilized a range of various model approaches to optimize where possible. The scoring and modeling

2. SUPERVISED LEARNING

2.1 Motivation

In our approach to explore this novel theory of lexical complexity and how it might aid classification, we used an algorithm outlined in the research paper "Estimation of Lexical Complexity using Language Semantics" by Nimavat and Champaneria published in 2017 to derive a new "Complexity Score" for each sentence. The details of the algorithm are outlined below in Section 2.3a. Although experimental, our hope was that this feature alone would be so informative that it would result in fair-performing supervised models even without significant tuning. Although trying a handful of base models, we decided to spend the majority of our model tuning efforts on Random Forest Classification and AdaBoost Classification using a Decision Tree Classifier as the base estimator.

2.2 Data Source

The training dataset used in this project consists of over 400,000 sentences from Wikipedia articles. Each sentence also has an assigned label of 0 or 1, denoting whether the sentence has been deemed "too complex" and requires simplification for a simpler and easier-to-read version of Wikipedia and is currently available on Kaggle.

We spent considerable time and effort exploring various metrics to measure lexical complexity and then performing intensive feature engineering to calculate a complexity score across the large dataset. For this we also relied heavily on the WordNet English dictionary. Using this dictionary, we built a JSON asset file containing all words found in the dataset along with their complexity score resulting in 136,542 unique words. Many obscure words and names were not found in the WordNet dictionary (e.g. zuleyka, zvori, zwalm) and contain complexity scores of zero in this asset file. Later when training models, we decided to assign all words with scores of zero the mean complexity score of all non-zero words. We were careful not to send multiple requests to WordNet for the same words and therefore are comfortable concluding that we sent roughly 136,542 API requests to WordNet to build this asset. The script to build this asset took more than 60 hours to complete but using this JSON, the time to calculate complexity scores of a sentence was reduced from 12 seconds on average down to under a tenth of a second, solving a serious potential feature engineering bottleneck in production.

2.3 Methods and Evaluation

2.3a Novel Feature: Lexical Complexity Score

To implement our Lexical Complexity Score, we replicated an algorithm outlined in the 2017 research paper titled "Estimation of Lexical Complexity using Language Semantics" by Nimavat and Champaneria.

Below is the algorithm proposed in the paper:

```
Algorithm: Semantic Word Complexity
Input: Word
Output: Complexity score depicting the complexity of the given word
1:  complexity_score = 0
2:  defn = get_definition(word)
3:  tokens = word_tokenize(defn)
4:  useful_words = remove_stopwords(tokens)
5:  for elem in useful_words:
        if elem in basic_word_list:
            complexity_score += 1
        else:
            defn2 = get_definition(elem)
            tokens2 = word_tokenize(defn2)
            useful_words = remove_stopwords(tokens)
            complexity_score += len(useful_words)
 6: return complexity_score
```

Key Components

The algorithm proposed above is highly dependent on three components and adjustments to any of them may result in dramatic changes to the ultimate complexity score assigned to a sentence.

The Dictionary: Some dictionaries favor complicated, robust, and verbose definitions. Our initial notion is that simpler words which evolved earlier in a language's lifetime are used as building blocks to define words which evolved later. Modern dictionaries strive for accuracy and in doing so may use modern words to define much more primitive ones. We still hope however, that our general assumption that easier words form the basis for more complex ones holds true in this case.

Because of the significant volume of the dataset, we opted to use the Python library PyDictionary (version 2.0.1) for speed and ease of use. We explored other dictionary APIs, but had concerns regarding rate and daily request limitations. Definitions provided through this library are sourced from WordNet.

The Basic Word List: This list contains the most basic English words which all other words are built upon. In the algorithm, any word contained in this list receives only a +1 to the complexity score, which is extremely little considering the average complexity score of all words in the entire dataset is 18. Increasing the size of this list would lower the mean assigned complexity score while removing words from this list would yield the opposite effect.

Although exploring a few additional basic word lists, we opted to use the combination of the two recommended word lists in the research paper. OgDen's 2000 word list and 1000 basic words from Wikipedia. A little web scraping and cleaning allowed us to create a set of strings to use.

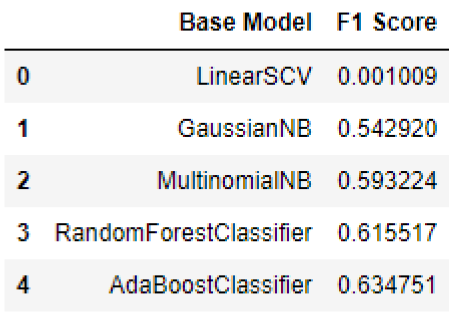
Stop Words: Because stop words were filtered entirely in the Lexical Complexity Algorithm, increasing or decreasing the number of stop words would impact the final mean complexity score assigned. We decided to use the standard NLTK (version 3.6.3) stop word list of 127 words.

Feature Engineering

In the research paper, the authors proposed summing the total complexity score of each word in the sentence. In addition to this metric, we decided to experiment by taking the average complexity score to see if it might yield better results. In the end, the sum of scores performed better after some further reflection; we believe we now understand why. By taking the average, we disregard the length of the sentence; however, it is probably a reasonable general assumption that the longer a sentence is, the more complex it is. Although this rule is not always true, statistically speaking the more words, the more room for confusion, misunderstanding, or encountering an unknown word. By taking the sum complexity score, longer sentences are much more likely to have a higher complexity score than shorter sentences which was not the case for the mean complexity score metric.

2.3b Rational for Random Forest and AdaBoost Classifiers

As a general benchmark, we initially explored feasible supervised models by exploring five different models. We trained them on all features using default parameters with the intention of taking the top two performers into a more robust tuning phase. Below are the initial F1-scores achieved for each of the five models.



	Base Model	F1 Score
0	LinearSCV	0.001009
1	GaussianNB	0.542920
2	MultinomialNB	0.593224
3	RandomForestClassifier	0.615517
4	AdaBoostClassifier	0.634751

Because the base Random Forest and AdaBoost Classifiers were the only models to achieve an F1-score over 0.6, we made the decision to move forward in those models with more aggressive randomized and grid searching of a wide array of possible parameter combinations.

2.3c Random Forest Classification

After tuning the number of estimators, max features, max depth, min samples split, min samples leaf, and bootstrap parameters, the final F1-score achieved by our model was 0.66 with accuracy of 0.6408 using the following parameters:

n_estimators=200
min_samples_split=10
min_samples_leaf=4
max_features='sqrt'
max_depth=20
bootstrap=True

Overall with only 6 features (a significant portion of them highly correlated with sentence length as discovered by our unsupervised model), although not as high as we had hoped, we feel this is a decent baseline accuracy for our model considering it relies essentially only on sentence length and our derived complexity score feature. Although we were able to tune the model for a 0.0444 increase in F1-score, which is not nothing, we did not find this increase particularly exceptional at the threshold of mid-sixties. Our model has roughly a 0.05 margin in F1-Score above and below the base score of 0.6155, making it affected by parameter choice but not overly dependent on it.

2.3d AdaBoost Classification

Without any tuning, this model showed the most promise in our initial exploration and after significant random and grid searching of many possible parameter combinations, the optimal combination we found resulted in an F1-score of 0.6762 and an accuracy score of 0.6005. We used a Decision Tree Classifier as our base estimator and the final parameters after tuning were:

base_estimator__max_depth=2
base_estimator__min_samples_leaf=5

n_estimators=50
learning_rate=0.001

We found the difference of nearly 0.08 in F1-score and the accuracy score to be intriguing, prompting us to investigate precision and recall scores which are 0.833 and 0.569 respectively. In other words, when this model predicts a sentence is complex, it is correct 83.3% of the time while it only correctly identifies 56.9% of all complex sentences.

2.3d PCA Components as Features

In our unsupervised learning PCA, we managed to capture more than 90% of the variance of the features in only two principal components. Because of this success, we decided to experiment using only these two principal components as features to a supervised model. We stuck with the Random Forest and AdaBoost models to better compare the success of the principal components to the full feature set. Without any tuning, the Random Forest classifier had an F1-score of 0.6316 and AdaBoost achieved an F1-score of 0.6609. Training on these two principal components actually had greater initial success than our full feature set.

2.4 Failure Analysis

I believe given more time, the final results could be vastly improved simply by switching the dictionary used. At times in the WordNet dictionary, the definitions of seemingly simple words are extremely complicated. An example of a failed classification, deemed too difficult by our model is the following sentence:

“A watering can is a container that is used to water plants by hand.”

Overall this sentence does not appear to be very complex. Exploring how it was broken down and processed by our Lexical Complexity Algorithm, we see that all tokens are filtered out except for the following six: watering, container, used, water, plants, and hand. So far everything looks to be in order. Let’s explore the derived complexity scores for each of the tokens, bearing in mind that the average complexity score for the entire dataset is roughly 18:

watering:	12
container:	5
used:	No definition for this exact token found (later assigned average of ~18)
water:	73
plants:	26
hand:	21

Water’s complexity score is extremely high and after investigating the definition returned by the dictionary, the cause is apparent:

> “binary compound that occurs at room temperature as a clear colorless odorless tasteless liquid; freezes into ice below 0 degrees centigrade and boils above 100 degrees centigrade; widely used as a solvent”

Likewise in this same example, another failure of this approach is highlighted. It is difficult to determine which definition should be used when many words have multiple definitions of varying complexity. In this case, the token “plants” used the incorrect definition “buildings for carrying on industrial labor” for this context. Perhaps there is an existing text-based tool to identify the most likely definition of a word, considering its context.

3. UNSUPERVISED LEARNING

3.1 Motivation

As with most unsupervised learning approaches, we wanted to utilize techniques which would allow us to identify possible underlying, distinguishing structure within the lexical complexity data. Given that our features include various measures related to types of word count as well as our complexity score (both sum and mean), it seemed likely to identify underlying principal components.

By performing PCA with our modified data set, we hoped to reduce our dataset to the smallest number of features which still account for most of our data’s variance. With KMeans, we wanted to find the most optimal number of clusters within our data if clusters existed.

3.2 Data Source

The data for our unsupervised models utilized the same source as our supervised models (**Section 2.2**) Our unsupervised approach allowed for the use of our principal components to train learning models apart from our main data source.

3.3 Unsupervised Learning Methods

Our unsupervised learning code followed a standard workflow procedure to best utilize our prepared data. Our unlabeled data was separated into its own dataframe and normalized prior to fit and transform with our models. Once the models were prepared, visualizations were used to aid in the interpretation of our data.

3.3a Principal Components Analysis

We began with the use of Principal Components Analysis for our first unsupervised model with the goals of reducing dimensionality of our data for further use and examining the importance of our features in relation to any principal components identified in the process. We first ran our PCA model to examine up to five principal components, given our five data features as an upper limit.

Once the model was created, we were able to examine the explained variance ratios for each component. Our first two principal components accounted for roughly 70.7% and 23% of the variance, respectively. Given that these two components accounted for over 90% of the data's variance, we were comfortable with focusing on those two components as we examined patterns within the data.

3.3b KMeans Clustering

In addition to PCA, we also used KMeans clustering to attempt to separate the data into distinct underlying clusters. Given that our data is separated into complex and non-complex sentences, we would hope to find significant clustering between at least two groups. The data was normalized with MinMaxScaler from sklearn prior to fitting within our Kmeans model.

To determine the optimal number of clusters in this method, we utilized Calinski-Harabasz and Davies-Bouldin scores, as well as the elbow method with the sum of squared errors for a range of cluster numbers.

3.4 Unsupervised Model Evaluation

3.4a Relationships and Insights

When looking at the relationship between our principal components and our data features a few details stand out, visualized in this biplot:

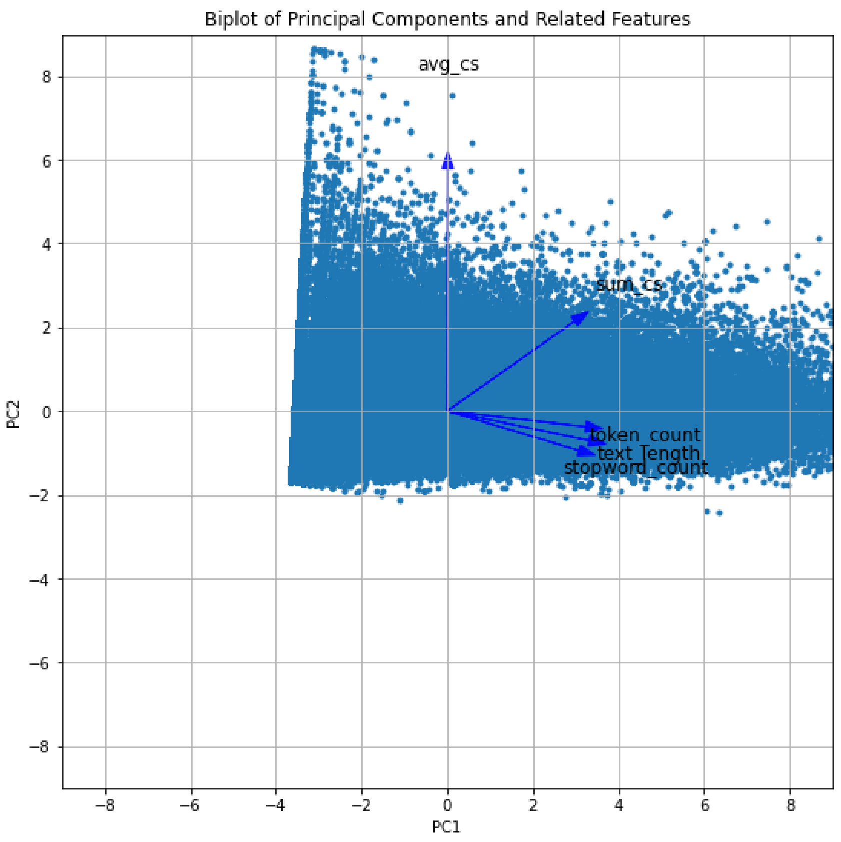


Figure 1. Biplot showing the principal component scores and feature loadings.

Four of our features show a strong relationship with our first principal component which makes up 70.7% of our variance: sum_cs, token_count, text_length, and stopword_count. These features are all measures of a sentence's length in their own way, so it makes intuitive sense for them to appear related to one another. Our second principal component (23% of variance) shows a mild relationship to our complexity score sum, but a strong relationship to the average complexity score feature.

To determine the most optimal number of clusters for our KMeans unsupervised model, we decided to utilize a few different approaches for a wide range of evaluation. If we first look at the model with the elbow method of sum of squared errors, we can see that the inertia starts to flatten out around 2 or 3 clusters. To take an even closer look, we were able to check the Calinski-Harabasz index scores for the ratio of between-cluster dispersion and within-cluster dispersion as well as the Davies-Bouldin index scores for the intracluster variances over the cluster centroid distance. Optimal clustering is typically found with high Calinski-Harabasz and low Davies-Bouldin; here it appears that 2 clusters provide the most optimal results.

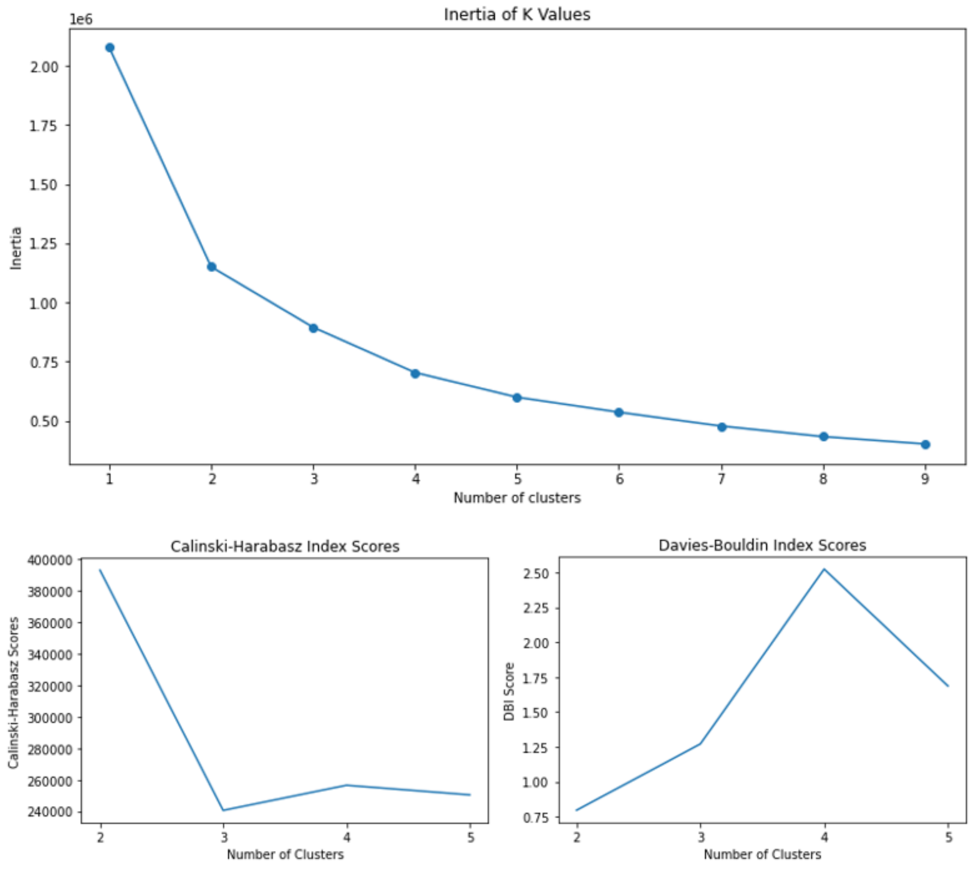


Figure 2. Charts comparing our KMeans model's inertia values, Calinski-Harabasz Index scores, and Davies-Bouldin Index scores to help identify the optimal number of clusters within unsupervised data.

3.4b Challenges in Analysis

When plotting our unsupervised models, visual analysis appears to show no real separation between possible clusters in the data. K-Means++ might help to provide for more predictable clusters given that it works to maximize the distance between centroids when initializing our clusters, but there still does not appear to be a specific demarcation between clusters. Also, there appears to be a lower limit to our first primary component, shown by what looks like a flat line to the left of our biplot's scatter points, which has the potential to skew other analysis.

4. DISCUSSION

4.1 Complexity Feature Engineering and Supervised Learning

Rather than focusing only on training the best model, we tried to address an engineering problem that would hinder production. Our complexity score feature originally required 12 seconds on average to calculate for a sentence. With the expectation of providing classifications in real-time, 12 seconds was far too long and we needed to build some sort of asset to dramatically reduce our feature engineering step. Considering the whole ML pipeline, we created an all_tokens.json asset file with the complexity score of every word in our lexicon. It is also designed in a way to efficiently update itself whenever new words are encountered and processed. Now on average, this feature is now calculated in less than a tenth of a second. We felt it was important to address this bottleneck first because no matter how accurate the final predictions, they would take too long to provide real value.

In terms of the complexity score feature engineering, we should have opted to use an English dictionary for children or non-native English learners. Returning to our water example, the Oxford Learner's Dictionary defines water as "a liquid without colour, smell or taste that falls as rain, is in lakes, rivers, and seas, and is used for drinking, washing, etc." which is clearly simpler than the definition of our dictionary provided in Section 2.4. Although we could not apply stemming or lemmatization to the words prior to looking them up in the dictionary, we should have applied some form of processing when checking if they exist in our basic word list.

4.2 Unsupervised Learning

In approaching real world data with the applications we have practiced throughout the program so far, the complexities of unsupervised learning became more apparent. Plotting your data will not be enough to have a solid idea of possible underlying structure, and we need to practice deeper analysis through various scoring and testing methods. One pleasant surprise in our use of PCA for dimensionality reduction came in the fact that we saw slight improvements in supervised learning models given our PCA output data.

With more time and resources, we would have liked to have utilized other clustering models to examine our data's structure. For example, we considered using Kernel PCA as an evaluation method, but ran into memory issues that prevented further study. Given the exploratory nature of unsupervised analysis, exploring any method possible would provide possible insight where our current models might have fallen short.

4.3 Ethical Issues with Supervised Learning

Wikipedia articles mixing foreign language words in with English not found in our dictionary would likely skew the final classification; we attempted to address this issue by assigning any unknown words the mean complexity score of all 136,542 words in our lexicon.

Another ethical consideration comes in the use of our pre-labeled training data. There is always the chance of mislabeled data, or perhaps different approaches to what the initial rating might have considered complex versus not. Even if a model provides favorable results, we must always be aware of the possibility of human error and allow for all possible avenues of evaluation.

4.4 Ethical Issues with Unsupervised Learning

As with any problem that involves human input and interpretation, there is always a chance for bias within results that might not be initially apparent. In this case with the decision of sentence complexity, complexity is a difficult concept to narrow down to the most common use case. Any number of factors can fall into a user's interpretation of a sentence or topic as complex, such as a user's language proficiency. Depending on how a model is trained, sentences and articles rated as not complex to a native English speaker might still be effectively very complex for other users whose primary language is not English. In our case of unsupervised classification, we already know that our complexity score feature has difficulties with non-English words and characters and might not serve as completely accurate in such cases.

Another area of potential bias is in that of model tuning. Our model interpretation showed that the use of two clusters might be most appropriate, but what if our interpretation was colored by the fact that this is a binary classification problem? If we already assume that there should be two and only two clusters, we might miss more subtle division of the data. Interpretation should utilize a wide range of testing and analysis wherever possible to minimize possible bias we might bring to the problem.

5. Statement of Work

Dylan worked on the lexical complexity score feature engineering, all_tokens.json asset, and tuning supervised models. Nicholas mainly worked with unsupervised model creation and analysis, as well as some initial supervised model approaches. Both team members wrote and edited their respective sections of the final report.