Unsupervised & Supervised Learning Approaches to Predict Text Difficulty

University of Michigan
School of Information
Master of Applied Data Science (MADS)
SIADS 696: Milestone II

Team 11
Dave Boudia, Josh Horner and Tim Strebel

Part A: Supervised Learning

Our project is an exploration of how Data Science can be used to identify text communications that are difficult or challenging to interpret. Most communication courses will emphasize that in the transmission of messages, the burden of understanding falls primarily upon the sender. Leveraging machine learning to determine what text is difficult to understand and what similarities and differences exist in both easy to understand and difficult text could have many important applications and consequences across several domains. One example we can cite is from medicine where literacy has major implications for patients. The standard reading level for patient consent forms is usually at the 6th grade level. If a provider creates a patient consent form that is above this reading level, they expose a certain cohort of patients to a precarious situation where they may consent to undergo a procedure in which they are not aware of all the implications and risks. Another example might entail ensuring that instruction manuals for appliances or complex machinery are written in such a way that they reduce the potential for misunderstanding, avoiding potential long-term or life-threatening injury. To attempt to mitigate this problem, our team has explored the possibility of utilizing machine learning to classify sentences that are difficult to understand.

Data Sources

The primary dataset we used to train our classifier is a corpus of sentences from the University of Michigan Predicting Text Difficulty Challenge on Kaggle[1]. The sentences came from two sources: standard Wikipedia content and simplified Wikipedia, which is “written mostly in a basic level of English[2].” The Training and Test datasets were sentences scraped from a variety of wikipedia pages from both versions of Wikipedia. The target variable 1 indicates that the sentence came from standard Wikipedia and needs simplification, and the target 0 indicates that it came from simplified Wikipedia and has already been simplified. The entire file on Kaggle came with the Wikipedia training and test datasets (48.5 MB and 14.6 MB respectively) in .csv format, an example submission for the competition (1.03 MB) in .csv format, and three additional readability resources.

The readability resources included:

- The Dale-Chall 3000 Word List (18.3 MB in .txt format) contains a list of words that, according to Readability.com[3], "are typically familiar to 80% of American 4th grade students (in the 90s)".
- The Brysbaert et. al[4] Concreteness ratings for 40,000 english lemmas (1.53 MB in .txt format). The concreteness scores were obtained from a study where the researchers recruited participants from Amazon’s Mechanical Turk to rate each word on a scale from 1-abstract to 5-concrete.
- The Kuperman et. al[5] Age of Acquisition (AoA) norms for over 50,000 english words (3.36 MB in .csv format). The AoA contains the approximate ages of when a word was learned for over fifty-one thousand words per their research study. Each row of the AoA file is a word with attributes such as alternative spellings, it's dominant part of speech, the number of letters it contains, syllables, the lemma version of the word, and an

---

1: https://www.kaggle.com/c/umich-siads-696-f22-predicting-text-difficulty
2: https://simple.wikipedia.org/wiki/Simple_English_Wikipedia
3: https://www.readabilityformulas.com/articles/dale-chall-readability-word-list.php
4: http://crr.ugent.be/papers/Brysbaert_Warriner_Kuperman_BRM_Concreteness_ratings.pdf
5: http://crr.ugent.be/archives/806

estimated age that the word is learned and a percentage of individuals who knew the word from the original study.

Feature Representations

Once we acquired our data, we began our classification task by conducting exploratory data analysis (EDA) to inform our data cleaning and feature engineering processes. We started by analyzing the frequency and pattern of missing data (missingness) in both Wikipedia files and readability resource files. We didn't find anything too alarming but noticed that the Dale Chall 3000 word list only contained 2,950 words, the concreteness ratings were missing one word, and the AoA was missing the estimated AoA for 19 words. After exploring missingness in our data, we began comparing statistics of the text corpus such as most frequent words, term-frequency inverse-document frequency (TF IDF) weights across both target variables using a 1 to 3 n-gram range. We did this analysis with both the raw words and lemmatized form of the corpus. At this point we began to notice opportunities to improve the quality and integrity of the Wikipedia texts.

Our initial finding was sporadic punctuation that was oftentimes repetitive and symbols such as umlauts that seemed to be a result of decoding errors during text extraction from the web. Our team felt we could improve our classification accuracy by removing these extraneous characters. Another thing we noticed were units of measurement that had inconsistent presentations. A couple of examples of this were "° C", "%", and "kg." To preserve these items and reduce variation in their representations, we used regular expressions to replace them with verbose representations: "degrees celsius," "percent," and "kilogram." The final thing we noticed was that the corpus contained tokens the tokens "-LRB-" and "-RRB-" which represented left and right brackets and "-NDASH-" and "-MDASH-" tokens to represent en-dash and em-dash. After some careful consideration, we decided to leave these tokens alone to prevent the Scikit Learn text vectorizers from removing them since they may provide some signal to our models. After these discoveries, we proceeded to create our text preprocessing pipeline.

Our team attempted to create a preprocessing pipeline solely relying on Python standard packages and Pandas. After experiencing exhorbitant processing times, we decided to move to PySpark to address the large amount of data and processing steps needed. Spark SQL provided an easy interface for performing numerous regular expressions to replace/remove unwanted characters and create standard conventions. In addition to cleaning the text, we were able to leverage an open source Spark NLP package developed by John Snow Labs to quickly lemmatize our text. The result was a preprocessing pipeline that applied 9 regex replace operations and lemmatized the entire 416,000 row corpus in about 3 minutes.

At this point we decided to implement and explore readability features. We started by implementing the Dale-Chall readability formula[6] to each sentence (see below). This is essentially a weighted sum of the percentage of difficult words and average sentence length. The difficult words represent words that are not found in the dale-chall list.

$$0.1579\left(\frac{difficult\ words}{words}\right) + 0.0496\left(\frac{words}{sentences}\right)$$

6: https://en.wikipedia.org/wiki/Dale%E2%80%93Chall_readability_formula

For the Brysbaert et. al Concreteness ratings, we created a sentence vector by looking up concreteness ratings for each lemma using a dictionary lookup. We then calculated the min, mean, and max concreteness values of the sentence vector. We engineered similar feature representations using the AoA resource creating a vector and computing the min, mean, and max estimated AoA of the sentence. Our final readability feature was to count the number of lemmatized words in each sentence as a feature representation. The simple idea for feature engineering ended up exceeding our expectations.

Once we finished engineering our readability features, we went back and incorporated these transformations into our preprocessing pipeline. Fortunately, the added steps did not add noticeable time to the preprocessing.

After computing our readability features, we explored the statistical characteristics of them using histogram and box pair-plots to observe the differences in distributions between the different classes (i.e., 0 and 1) defined by our target variable. It was interesting to discover that most of the readability features we engineered showed little significant difference in their interclass distributions. The readability features that showed the most significant differences in their interclass distributions were: the **Dale Chall score**, **max AoA**, and the **number of lemmas**.

Methods & Evaluation

After cleaning our data and coming up with a set of feature representations, we came up with a plan to analyze the classification results of 7 different models. The best performing model would be put forth in the Kaggle challenge. Classifiers analyzed included: logistic regression, linear support vector classifier (linear SVC), multinomial naive bayes, decision trees, random forest and XGBoost random forest. For good measure, we trained a dummy classifier as a baseline for our other models. Each classifier was trained using the same 90-10 train-test-split data generated from our preprocessing pipeline. Our team decided to use the same train-test-split for two reasons 1) to ensure our results were consistent amongst different models (since we divided the work) and, 2) to ensure we were all using a held-out test dataset to test for overfitting and data leakage in our classification pipelines.

After dividing the work three ways, we each built machine learning pipelines for our respective models with some ground rules. The rules were that we could use any hyperparameters, pipeline components, and additional preprocessing steps we wanted as long as they were contained within a single callable object.

For the most part, we used Scikit Learn models and pipeline components. For the linear models, Incorporating additional preprocessing methods such as data imputation was necessary since there were a few missing values in some of the readability features. To vectorize our corpus, we tried both the count vectorizer and term-frequency inverse-document frequency (TF IDF) vectorizers on our cleaned and lemmatized corpus. We tried different n-gram ranges with both vectorizers to see which performed best.

To combine our entire feature set (readability features + text vectors), we used the Scikit Learn column transformer. Because we were combining sparse and dense vectors, our team found it

---

7: https://www.cs.cornell.edu/people/tj/

advantageous under most circumstances to incorporate a binning pipeline component for the dense readability features. This helped significantly speed training convergence.

For hyperparameter tuning, our group employed stratified grid-search cross-validation using 10 folds, comparing mean accuracy scores amongst hyperparameter combinations. We identified the importance of observing both training and test scores vigilantly to identify cases of overfitting. After multiple iterations of grid search, the best combinations of parameters and features for each classifier were selected and a final model was trained and scored on a held out test set. When training the XGBoost classifier, we wanted to incorporate the XGBoost API[9] so that we could implement stop-round training. Stop rounds helped us prevent overfitting by controlling the number of trees added to our ensemble by stopping the boosting process after a predetermined number of rounds (25) of no improvement to accuracy. To inform the number of estimators (trees), we used the best-performing round. Due to the nature of using TF IDF as a pipeline component, we had to create our own grid search stratified 10-fold cross validation evaluation process by combining Scikit Learn and XGBoost API components to avoid data leakage.

For classifier evaluation, we chose several metrics to obtain a broad perspective on our model performance and to look for any significant trade-offs. Given that our data exhibited a perfect 50/50 class balance, we chose the area under the receiver-operator characteristics curve (ROC AUC) as our primary metric. Since the Kaggle competition used accuracy as the primary evaluation metric, we thought it would be useful to include that to get a sense for how our final classifier would perform. Finally, we included the F1 score to see how well each classifier balances precision and recall. The results from our modeling efforts are shown in Table 1.

Model	ROC AUC (Mean)	ROC AUC (95 % CI)	Accuracy (Mean)	Accuracy (95 % CI)	F1 (Mean)	F1 (95 % CI)
Dummy Classifier	0.501	(0.499, 0.503)	0.500	(0.499, 0.501)	0.500	(0.499, 0.501)
Decision Tree	0.722	(0.721, 0.723)	0.659	(0.658, 0.660)	0.684	(0.683, 0.685)
XGBoost RF	0.747	(0.746, 0.748)	0.676	(0.675, 0.677)	0.686	(0.684, 0.688)
Random Forest	0.750	(0.749, 0.751)	0.679	(0.678, 0.680)	0.694	(0.693, 0.695)
Logistic Regression	0.772	(0.771, 0.773)	0.701	(0.700, 0.702)	0.704	(0.703, 0.705)
Naive Bayes	0.785	(0.780, 0.790)	0.708	(0.704, 0.712)	0.716	(0.712, 0.720)
Linear SVC	0.808	(0.806, 0.810)	0.732	(0.731, 0.733)	0.740	(0.738, 0.742)

Table 1: 10-fold Stratified Cross-validation Results for All 7 Supervised Models

It wasn't surprising that our linear SVC was our best performing model. Our team discovered through additional sources that linear SVC's tend to perform well at text classification for many reasons. Document vectors often represent a high-dimensional space, text often has relatively few irrelevant features, document vectors are sparse containing mostly zeros, and most text classification problems are linearly separable[7]. Linear SVC determines class labels by creating "decision-boundaries" that separate classes controlled by the complexity hyperparameter, "C",

with lower values creating larger boundaries. Our classifier had a moderately large decision-boundary (C=0.5) and employed "L1" (Lasso) regularization, reducing the features down to only the most important for the classification task.

One of the models that did not meet our expectations was the XGBoost random forest classifier. While none of the decision tree-based models performed well, our team thought that XGBoost would at least put up defensible performance on this task. We think this is due to the stochastic nature of XGBoost (training on subsets of the data) in the face of the inductive nature of text classification.

Feature Importance and Ablation Analysis

Figure 1 provides the top 15 features from the linear SVC derived from the model coefficients. Positive coefficient values represent features that contribute to difficult sentence classifications and negative values contribute to simplified classifications. This is interesting because you can start to get a sense about how simplified Wikipedia works. We can see that simplified articles tend to contain fewer en-dashes and use more commonly understood words. Instead of “footballer,” and “Municipality,” we could say “football player,” and “city.”

Figure 1: Linear SVC Top 15 Most Important Features

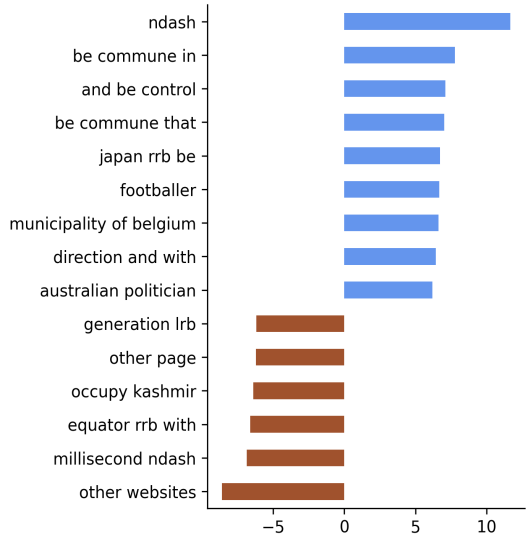
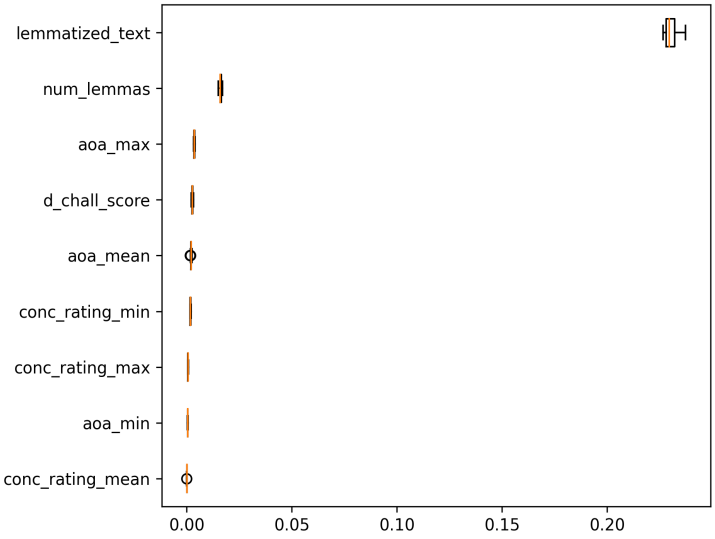


Figure 2: Linear SVC Feature Permutation Importances



We conducted a ranked feature permutation analysis on the linear svc model (Figure 2) using a held-out test set and ROC AUC as our primary evaluation metric. This method attempts to isolate and shuffle data for each feature while holding the other features constant to identify the impact to classification performance. The results show the most important feature (not surprisingly) is the lemmatized text. A surprising second-place is the number of lemmas contained within the sentence. These findings are inline with our feature importance plot and which doesn’t contain any of the readability features. This suggests that the words and phrases themselves are most conducive to the prediction task.

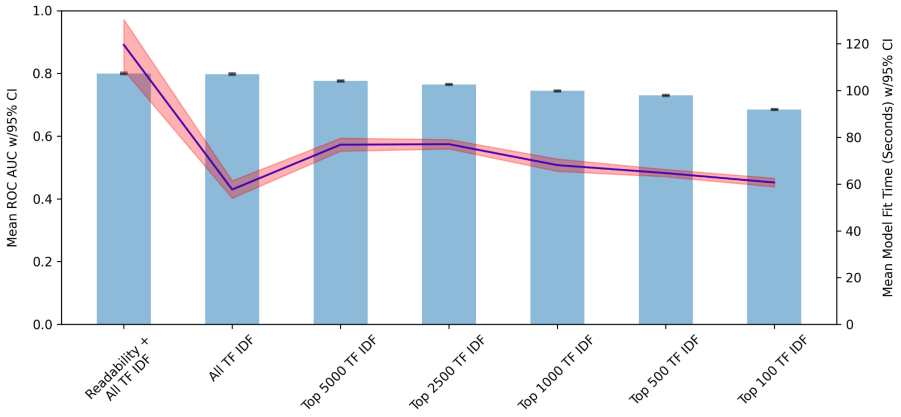
To explore the impact of removing features from our dataset and to see what tradeoffs we could achieve in training times versus model accuracy, we set up a feature ablation experiment (see table 2). It is worth noting that the totality of our TF IDF features is approximately ~29,000 before we began pruning them. After we dropped the readability features, we noticed a trivial difference in mean ROC AUC score. As we begin to trim TF IDF features however, we begin to notice significant declines in performance.

Feature Set	ROC AUC (Mean)	ROC AUC (95 % CI)	Accuracy (Mean)	Accuracy (95 % CI)	F1 (Mean)	F1 (95 % CI)
Readability + All TF IDF	0.800	(0.798, 0.802)	0.723	(0.721, 0.725)	0.722	(0.720, 0.724)
All TF IDF Features	0.797	(0.795, 0.799)	0.721	(0.719, 0.723)	0.720	(0.717, 0.723)
Top 5000 TF IDF Features	0.776	(0.775, 0.777)	0.703	(0.702, 0.704)	0.702	(0.700, 0.704)
Top 2500 TF IDF Features	0.765	(0.764, 0.766)	0.694	(0.692, 0.696)	0.694	(0.692, 0.696)
Top 1000 TF IDF Features	0.744	(0.743, 0.745)	0.677	(0.676, 0.678)	0.679	(0.678, 0.680)
Top 500 TF IDF Features	0.730	(0.729, 0.731)	0.664	(0.663, 0.665)	0.670	(0.668, 0.672)
Top 100 TF IDF Features	0.685	(0.684, 0.686)	0.633	(0.631, 0.635)	0.645	(0.644, 0.646)

Table 2: Linear SVC Feature Ablation Results Over Stratified 10-fold Cross-validation

To further solidify this point, we plotted the mean fit-times of the feature sets against their performance to see exactly how much time we could expect to save by dropping features versus the cost in ROC AUC. What was surprising about this analysis is that fit times actually **increased** when we began to restrict the TF IDF vectorizer to the top 5000 features by weight. We think this might be due to additional computation tasks needed to rank-order the features in the Scikit Learn implementation (see figure 3).

Figure 3: Linear SVC Feature Ablation Trade Off Over Stratified 10-fold Cross-validation



Sensitivity Analysis

One aspect that was advantageous about the linear SVC is that there were relatively few hyperparameters to tune. To get a sense of how changes in key hyperparameters impacted classification accuracy, we set up an experiment to test how sensitive our linear SVC was to changes in complexity (C) and the max term-document frequency threshold (max_df) in our TF IDF vectorizer. Figures 4 & 5 below demonstrate the changes in mean ROC AUC scores using 10-fold stratified cross validation plotted over different values of these hyperparameters. We noticed significant changes in AUC scores in the face of changes in C. This is likely because C controls the amount of L1 regularization that is applied. When C is too low, the features become too sparse and when L1 is too high, not enough features are eliminated and the model suffers from overfitting due to the curse of dimensionality. We also notice that our model isn't particularly sensitive to changes in max_df and that we see steady performance gains as it approaches 0.6 (or -0.22 in log scale).

Figure 4: Linear SVC Sensitivity Analysis C

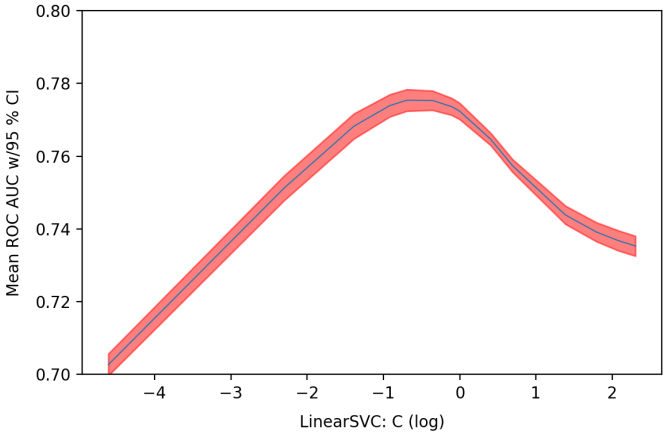
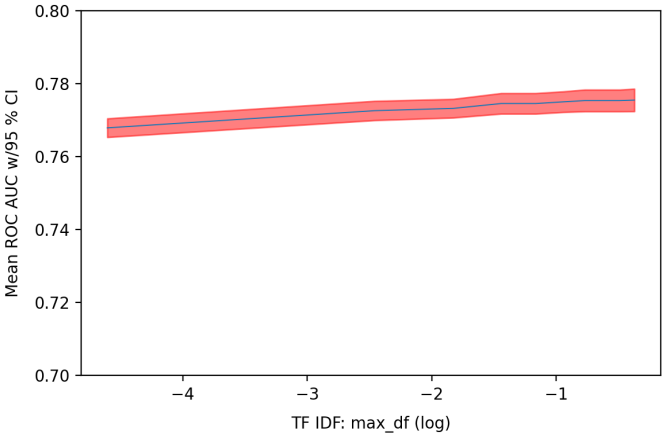
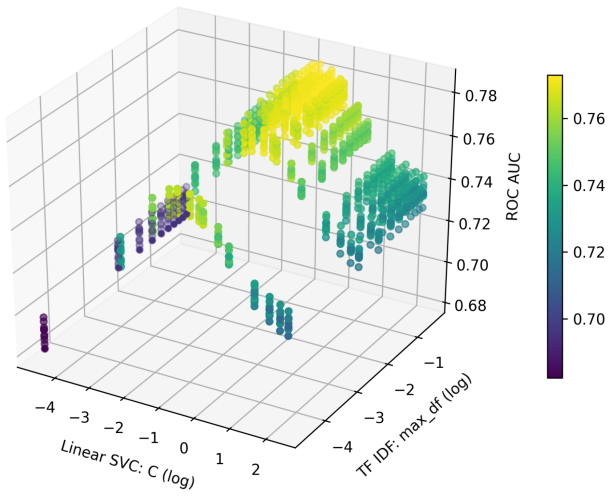


Figure 5: Linear SVC Sensitivity Analysis TF IDF max_df



When we compute and plot this analysis using 3-dimensions (figure 6), we see how changes in the two hyperparameters work together. We can see that optimal values of C and max_df are about 0.5 and 0.6 respectively. These values are approximate to what our grid-search cross-validation hyperparameter tuning yielded (0.5 and 0.5 respectively).

Figure 6: Linear SVC Sensitivity Analysis C vs TF IDF max_df

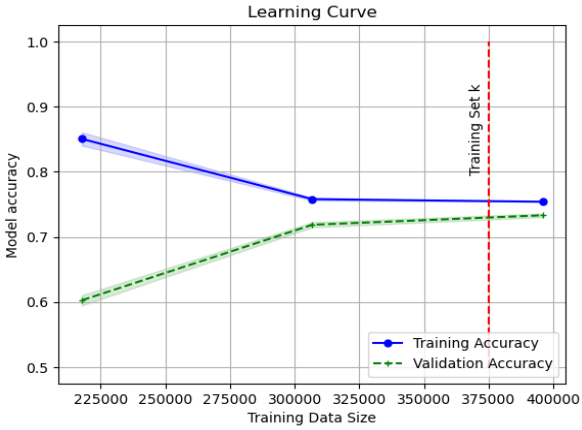


Learning Curve Analysis

To get a sense for how our linear SVC's classification performance was affected by the amount of training data it is given, we set up a learning curve experiment (Figure 7). In this experiment, we started with 200,000 records and iteratively increased the number of observations while recording the stratified 10-fold cross-validated accuracy scores. We see significant overfitting occurring for training sets with less than 300,000 observations. As we increase beyond 300,000 observations we start to notice a convergence in accuracy occurring for both training and testing data sets with the gap between the two decreasing more gradually.

Given the computational complexity of SVC when run on a high number of observations, it was difficult to power sufficient iterations to paint a high resolution picture of relative performance. Nonetheless, we can see that gains seem to taper just above 300,000 rows. Validation accuracy does not improve markedly after that threshold, even after adding in the previously held-out training set observations.

Figure 7: Linear SVC Learning Curve



Failure Analysis

After running thorough analyses on our model, we wanted to observe some anecdotal cases where our model was succeeding and failing to properly classify sentences. We ran our linear SVC model and compared predictions with ground-truth labels, evaluating examples of label agreement vs a sample of misclassified text. The mean number of lemmas for both the easy and difficult data were 13.5 and 24.6, respectively. Interestingly, we found that this trend reversed in our label-disagreement sample. In this sample, we observed a mean of 23 lemmas in the simplified sentences and 14 in the difficult ones.

Looking at individual observations, we noticed cases where easy sentences had ground-truth labels that were the opposite of what we would expect. For instance, “He served as Minister of Finance from 1993 to 2002 .” had a ground-truth label of difficult yet the linear SVC model classified as easy. Conversely, “Cosmic microwave background radiation , 3K blackbody radiation that fills the Universe Radiant energy, radiation emitted by a source into its environment ." was labeled as easy yet our best classifier suggested it was hard. We feel that this demonstrates an important dichotomy in how text difficulty was determined vs how our classifier understood difficulty: sentence structure/composition (e.g inclusion of numbers) versus the complexity of meaning.. Additionally, we found cases where the same sentence (document) or portion of a sentence was duplicated in the data set. In these cases, it's important to understand and evaluate the methods and heuristics involved in the labeling process.

In spite of questions about the validity of the truth labels, the classifier put importance on the number of lemmas as a feature. Generally, this makes intuitive sense as one would expect more difficult texts to be lengthier than easier texts. However, there were cases where phrases or non-sentences, such as html markup (e.g. *'align = " left " Atmosphere'*) were labeled as difficult to understand but the classifier predicted they were easier. Though the number of lemmas was the second most important feature, we saw that its exclusion wouldn't significantly downgrade classifier performance. It's possible that including evaluations on syllable length (mean syllable length) might mitigate this over reliance on quantity of words.

Final Model

After collating our analyses, we trained our final linear SVC pipeline which we submitted to the Kaggle competition. Table 3 represents our final model and hyperparameter settings. Our accuracy score for the competition was **0.734** and as of today, our team is in fourth place (out of 9) on the leaderboard.

Model	Penalty	C	TF IDF N-gram Range	TF IDF min_df	TF IDF max_df	Imputation*	K-bins*
Linear Support Vector Classifier	l1	0.5	(1, 3)	0	0.5	mean	quartiles

Table 3: Final Model and Hyperparameters

*Imputation represents how readability feature missing values were imputed.
*K-bins represents binning applied to readability features

Part B: Unsupervised Learning

We wanted to compare clustering and topic modeling techniques to determine whether either approach would form structure around the ground truth labels or whether either approach would expose latent features (e.g. topics) in the corpus that would identify text “difficulty.”

To do this, we employed the same processed corpus from our supervised pipeline. For topic modeling we used the tf-idf vectorizer and numeric features. For clustering we employed Spacy's pretrained large english vocabulary model to generate large dimensional space for sentence embeddings. Based on prior work, we felt sentence embeddings would yield interesting semantic relationships across the corpus. The large corpus generated 300 features (dimensions) with each feature generating a number. Each number is the average of a different linear combination of weights (from the pretrained model) generated from the sentence. Embeddings were generated during run-time based on sampling of the corpus.

Clustering

We evaluated three clustering algorithms: K-Means, Agglomerative and DBSCAN.

For k-means and agglomerative clustering we ran through a list of "k" cluster sizes, recording the mean silhouette score, homogeneity, completeness and davies-bouldin scores. In the case of DBSCAN, we chose different combinations of epsilon and minimum samples recording the same metrics as the other algorithms. Each cluster was plotted in a scatterplot for visual inspection of how well-defined the clusters were based on the given parameters. Parameters and metrics were recorded and ranked in Pandas. We relied on mean silhouette score as the primary metric of cluster quality evaluating how clusters formed in the absence of labels as well as provide us a sense of the "certainty" of cluster assignment. For additional insight, homogeneity (measuring "pureness" of cluster membership) and completeness (how much of each class was captured in their respective cluster).

K-means was chosen due to its general-purpose, straight-forward implementation and interpretable results. We found that it scaled well to large sample sizes without requiring substantial computing resources. Surprisingly, this algorithm performed the best among the three tested especially when preprocessing the embeddings using PCA resulting in 3 clusters with a mean silhouette score of .38. This score was still quite low implying that the resulting clusters were not tightly compacted as optimal scores would be closer to 1. We also didn't find natural clustering occurring based on text difficulty labels.

Figure 8: Silhouette Scores for K-Means

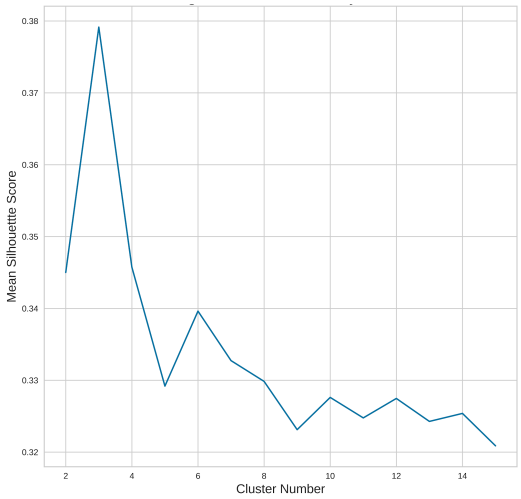


Figure 9: K-Means Clustering (k=3 using PCA)

Another algorithm we tried was Agglomerative Clustering with its bottom-up hierarchical approach. Ward's method proved to be the best cost function resulting in much cleaner clustering on our dataset vs single, average and weighted linkages from visual inspection of the resulting dendrograms (all using Euclidean Distance). This combination resulted in 2 clusters (among testing ranges between 2 and 15) with a silhouette score of .15. It should be noted that we tested with very large values for clusters up to 10000 clusters (which had the best mean silhouette score of .18 and the lowest davies-boudin score of .083). There were several important trade-offs using this clustering method. The first trade-off was that it was computationally expensive, constraining the number of observations we could provide to a mere fraction of what K-Means could handle in Google Colab. Second, given that each data point starts as its own cluster, there are a tremendous number of clusters generated before cluster sizes become manageable. We noticed that silhouette scores dropped significantly as cluster sizes moved from 2 clusters but then grew larger and exceeded scores produced by 2 clusters with k greater than 10000).

Figure 10: Agglomerative Clustering

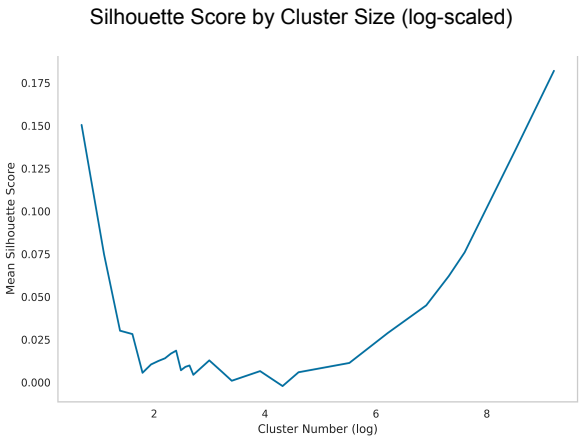
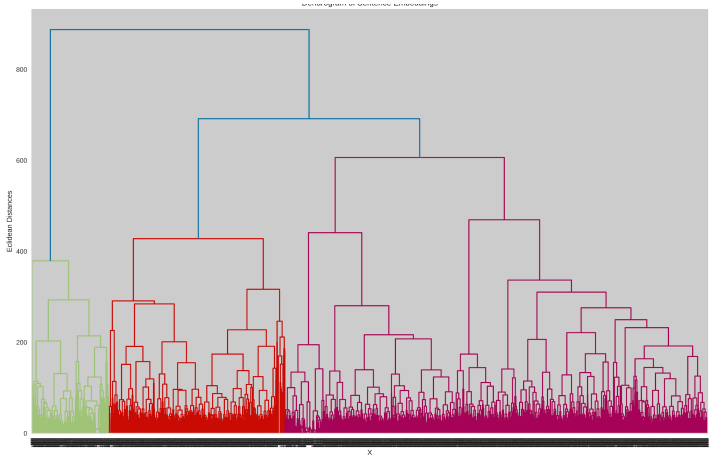


Figure 11: Agglomerative Clustering Dendrogram using Ward's Method



Finally, we ran DBSCAN as an alternative clustering method as it didn't require specifying cluster numbers ahead of time. It can also separate out "noise" (points with no cluster membership) from the resulting clusters. To tune DBSCAN, we evaluated several combinations of eps (representing maximum distance between two observations) and min_samples (number of samples in a neighborhood for a point to be considered a core point) parameters. Unfortunately, DBSCAN was the poorest performer based on mean silhouette score, generating very large k cluster values (18000+) and by extension creating very high homogeneity scores (given the small number of text for each cluster).

Regardless of the Clustering method, employing sentence embeddings made it more challenging to decipher which attributes of sentences led to the clustering results. This was partially mitigated by extracting the original sentences that were nearest (using Euclidean distance) to cluster centroids (via K-Means), applying sentences to the dendrograms (Agglomerative clustering) and/or applying labels as columns back the text dataframe and evaluating by membership. This still required using human evaluation to make sense of or extract out why sentences were clustered together. For Agglomerative Clustering there was an additional challenge of reducing the number of "p" levels truncating the dendrogram plot (i.e. reduce the number of visible clusters) when applying text labels to see cluster formation. Despite this, it still resulted in a very large plot/number of clusters to review. Finally, relying on a single evaluation metric (mean silhouette score) doesn't provide insight on the composition of the clusters. Therefore, employing a holistic approach to evaluation using several metrics, such as the homogeneity, completeness, davies-bouldin, visualization and manual inspection of clusters provide more clues into what led to the results.

Similar to the supervised learning portion, clusters appeared to form around sentence lengths (long sentences vs single words or phrases akin to number of lemmas). It also appeared that composition of characters had some importance, as accents, numbers (e.g. inclusion of dates) or symbols tended to be grouped together. As expected, it appears that sentences that shared identical matches or very similar words clustered together, which would make sense from a

Euclidean Distance perspective. However, it didn't appear, based on our metrics, that clustering was an effective method for either evaluating difficulty or extracting topics. It appeared that the clustering put more importance on sentence structure and the use of similar words/patterns than semantic meaning, despite the use of sentence embeddings as a feature choice. Table 3 shows closest sentences by cluster centroid for the highest scoring K-Means model. In the first cluster, we see several examples of sentences having different topics but referencing dates. In the second cluster we see references to accents and special symbols and in the third cluster, we see more cleanly formed, yet complex sentences.

Table 4: K-Means (k=3) Closest sentences by Centroid

Cluster	Top 5 Closest Sentences (by Euclidean Distance) to Centroid
1	Croatia national football team is the national football team of Croatia . Croatia national football team is the national football team of Croatia . The authoritarian King Carol II abdicated in 1940 , and succeeded by the National Legionary State , in which power was shared by Ion Antonescu and the Iron Guard . The district was created in 1974 by merging the two districts of Eschwege and Witzenhausen , which had both existed with only slight modifications since 1821 . The district was created in 1974 by merging the two districts of Eschwege and Witzenhausen , which had both existed with only slight modifications since 1821 . He makes cameo appearances : in both The Legend of Zelda : A Link to the Past and The Legend of Zelda : Ocarina of Time , Mario appears on a portrait , and in Metal Gear Solid : The Twin Snakes he appears as a small statue .
2	No. 2 -LRB- film -RRB- , a New Zealand feature film It is located at 34 Â 14 ' 0N 72 Â 1 ' 0E and has an altitude of 298m -LRB- 980 feet -RRB- . Ã ngel Luis SÃ nchez -LRB- born September 20 , 1983 in Humacao , Puerto Rico -RRB- is a Major League Baseball shortstop for the Houston Astros . Connecticut College Project 's leader .
3	It is located in Canberra , Australia 's capital city . A two-dimensional spiral may be described most easily using polar coordinates . There the radius r is a continuous monotonic function of angle Î -LRB- theta -RRB- . The county seat is Ellensburg , Washington . Widely regarded as a pinnacle in realist fiction , Tolstoy considered Anna Karenina his first true novel , when he came to consider War and Peace to be more than a novel . He has since written many novels and other works , and his fiction has been adapted into motion pictures , notably the Hellraiser series .

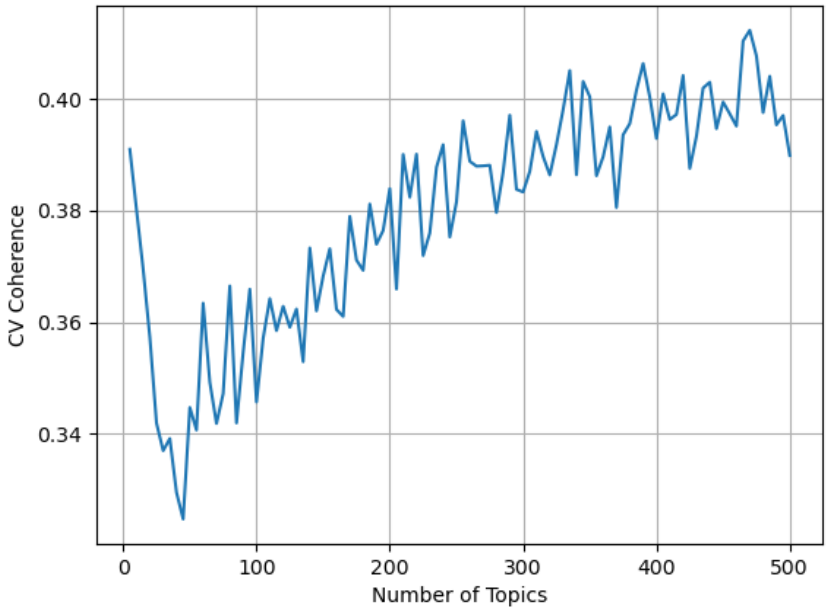
Topic Modeling

We attempted topic modeling, using Non-Negative Matrix Factorization, chosen because of its tendency to perform better on shorter documents. Since the sentences originate from Wikipedia, a medium already organized by topic, the intent was not to create a short list of human interpretable topics. Rather, we set out to explore and better understand the dataset while hopefully gaining insight into the original classification mechanism. One hypothesis was that the model might identify (as topics) some of the same characteristics that influenced a sentence's label. In addition, it is possible a downstream supervised model might be improved by grouping documents into topics before training. In other words, the topic might be a predictive factor in the sentence's labeling as either simple or difficult.

Before fitting the NMF model, we sought to determine the optimal number of topics using coherence scores calculated on the *CV* measure. As with this project's other analyses, resource constraints were a consideration. Thus, calculating coherence could only be feasibly accomplished for a sample of the data and/or a relatively narrow range of topic counts. Below

are the results based on a run for between 5 and 500 topics (increments of 5) on 5,000 rows. Coherence appears to be roughly proportional to the number of topics chosen, which makes sense given an understanding that the data are made up of pairings of sentences on the same topic. Along the same lines, we found that the classes are very balanced within topics. It seems that topic modeling cannot tell us much beyond what we already know. Namely, the data is composed of thousands of topics, evenly distributed between the labeled classes.

Figure 12: NMF - Coherence Scores by N Components



The topics generated were certainly picking up on some semantic relationships. A sample of 10-word topics reveals some interesting groupings and also shows that the classes remain balanced, even within these topics. Instead of separating the labels, it seems to have projected onto an orthogonal dimension that, while interesting, fails to help explain the sentence labels.

Topic	Simple 0	Difficult 1
east, people, south, west, northern, north, region, commune, department, france	9,845	10,329
family, music, world, large, group, number, people, area, work, include	13,115	15,694
film, large, form, series, work, game, world, american, good, know	5,800	8,256
music, country, number, small, english, write, mean, form, people, use	6,959	7,289
national, american, member, war, country, town, university, county, united, state	7,607	8,779
river, province, population, area, locate, district, new, county, large, city	11,833	13,131
september, april, march, december, mean, english, german, american, lrb, rrb	75,004	65,398
september, july, january, national, american, team, player, football, play, bear	20,620	22,287
series, world, work, new, film, small, large, people, game, make	6,047	4,058
war, people, old, work, start, world, late, new, time, year	30,716	32,325

Table 5: NMF - 10 topics x 10 highest weight terms

Discussion

Supervised Learning:

It was interesting to see the more simplistic classifiers achieved better performance than more complex classifiers. Even among the different options for Support Vector Machines, it was surprising to see the Linear model outperform its kernelized variations. This is a testament to the inductive nature of text classification where the classification target is usually weighted by many specific words and phrases. It is important that the classifier be able to account for as much of the vocabulary as possible for the sake of accuracy and to flexibly adapt for the addition of new vocabulary. Many of our more complex classifiers were stochastic, fitting only sub-sections of our data or were prone to overfit the sparse document vectors.

For future advancements of this work, we could see this problem being approached in a couple of different ways. Either using bigger models or better proxies for text difficulty. Unfortunately our readability features didn't perform all that well. Perhaps better feature engineering or further studies into readability markers could lead to the creation of continuous variables that can train classifiers to predict text difficult in far lower dimensions. Conversely, it is also conceivable that the development of a very large neural network for this task could account for an incredibly large vocabulary and contain a large number of tunable parameters that could perform well in English and even multiple languages.

Unsupervised Learning

In the course of applying unsupervised learning to our classification task, we learned that it is not necessarily suitable for our given objective and the data we are using. Our unsupervised results, on the other hand, serve to demonstrate the complexity of our data, rather than summarizing or simplifying its representation. This was still instructive, as it confirmed our initial impressions of the data as a diverse sample of myriad random topics, but ultimately it did not help much with our classification task.

In the case of both clustering and topic modeling, analysis revealed properties in the data that were already understood. The sentences vary widely by their topics and latent structure. It was interesting to see that K-Means, given its more simplistic approach to clustering, outperformed the other models (based solely on silhouette score) with a reasonable number of clusters (3), despite there appearing to be no semantic/topical similarity within the clusters (outside of the differences discussed earlier).

With more time and resources, it would be interesting to see how additional preprocessing methods for text (additional regex substitutions) would impact the clustering and topic modeling. Additionally, given more time, we may have chosen to incorporate Latent Dirichlet Allocation into our topic modeling analysis. NMF was chosen initially for its effectiveness with short documents and the interpretability of its topics. However, given more time we would have liked to be able to compare the two approaches.

Ethical Considerations

At the beginning of our report we outlined a couple of situations where literacy had potentially harmful implications for two different domains. If machine learning were to be used to classify sentence difficulty, it would be incumbent upon the data scientist to continually validate the results of the model on a robust test set from the domain of interest. The data scientists would also want to check for algorithmic bias where the model may not generalize to certain corpora that have a tendency to be read by specific groups defined by inherent characteristics.

Concerning ethics applied to our project, we have made our code and datasets publicly available on GitHub. We have given fair warning to potential users of the limitations of the data and model in the readme file. This applies to our supervised approaches more than our unsupervised approaches. We want to try to prevent the potentially harmful situation of someone using one of our classifiers in full-confidence in a potentially harmful real-world scenario.

Dave Boudia	Josh Horner	Tim Strebel
Part A: Supervised Learning		
● Baseline, decision tree & naive Bayes classifiers ● failure/error analysis	● Linear support vector classifier ● Learning curves	● Logistic regression, random forest, and XGBoost RF classifiers ● Collation of classifier results ● Feature ablation and importance analysis ● Sensitivity Analysis
Part B: Unsupervised Learning		
● K-means, agglomerative and DBScan Clustering	● Principal Component Analysis ● K-means Clustering ● Topic Modeling	● Research & technical support
Combined		
● Exploratory data analysis (EDA) ● Dale Chall ● ● Data preprocessing pipeline development (RegEx & readability implementation) ● Technical writing and editing	● Exploratory data analysis (EDA) ● Dale Chall & Number of Lemmas features ● AoA features ● Data preprocessing pipeline (RegEx & readability implementation) ● Technical writing and editing	● Exploratory data analysis (EDA) ● D-Chall formula & concreteness rating features ● Data preprocessing pipeline (PySpark & Spark-NLP implementation) ● Technical writing and editing

References

Brysbaert, M., Warriner, A. B., & Kuperman, V. (n.d.). Concreteness ratings for 40 thousand English word lemmas. Concreteness ratings for 40 thousand generally known English word lemmas. Retrieved October 23, 2022, from http://crr.ugent.be/papers/Brysbaert_Warriner_Kuperman_BRM_Concreteness_ratings.pdf

Joachims, T. (n.d.). Text Categorization with Support Vector Machines: Learning with Many Relevant Features. Home Page of Thorsten Joachims. Retrieved October 23, 2022, from https://www.cs.cornell.edu/people/tj/

Kuperman, V., & Stadthagen-Gonzalez, H. (n.d.). CRR " age-of-acquisition (AOA) norms for over 50 thousand English words. Center for Reading Research. Retrieved October 22, 2022, from http://crr.ugent.be/archives/806

Scott, B. (n.d.). The Dale-Chall 3,000 Word List for Readability Formulas. The dale-chall 3,000 word list for readability formulas. Retrieved October 22, 2022, from https://www.readabilityformulas.com/articles/dale-chall-readability-word-list.php

University of Michigan School of Information. (n.d.). UMICH SIADS 696 F22: Predicting text difficulty. Kaggle. Retrieved October 22, 2022, from https://www.kaggle.com/c/umich-siads-696-f22-predicting-text-difficulty

Wikimedia Foundation. (2022, February 8). Dale–Chall readability formula. Wikipedia. Retrieved October 22, 2022, from https://en.wikipedia.org/wiki/Dale%E2%80%93Chall_readability_formula

Wikimedia Foundation. (2022, October 1). Simple english wikipedia. Wikipedia. Retrieved October 22, 2022, from https://simple.wikipedia.org/wiki/Simple_English_Wikipedia