SIADS 694 & 695 Capstone Project

Authors: Holder Miller-Schaeffer and John Zupan
All code can be found on Github

Supervised Learning

Motivation

One of the most useful applications of machine learning is to remove or decrease the amount of repetitive, simplistic work placed upon humans. Removing these menial tasks can free people to work on more creative and interesting challenges. With that goal in mind, we aim to identify sentences in a corpus that are complex enough that they may need to be simplified. As described in this overview, there are many reasons a text may require the complexity of its content to be reduced, and this is an extremely time-consuming task for a human. Coincidentally, identifying these complex sentences is also a task very well suited for a machine learning algorithm. For every sentence that our algorithm recognizes as "already simple, no need for human review", real human time is saved. By confidently classifying sentences into their correct group we will provide a huge efficiency boost to the humans that would have previously had to read and review every sentence in the corpus. Of course, a human will still be required to simplify the sentences marked as "complex" (for now), but an extension of this work could be to design an algorithm that reduces the complexity of a sentence without changing the meaning in our next project.

Data Source

The data creation and collection for this task was very straightforward - all data used was acquired from Kaggle using their python library and API [1]. The training data set contained 416,768 records, while the evaluation set submitted to Kaggle contained 119,092 sentences. In addition to the raw sentences, Kaggle also provided supporting resources, which are described here, ranging in length from 3,000 records to ~50,000 records. All data was found in either *.csv* or *.txt* files and easily accessed using Pandas. In addition to the complete description of the training data found in the link above, Table 1 in the appendix has a full description of the 28 data fields used in our model. To supplement the data set, average word vectors (100-dimensions) for each sentence were calculated using a Word2Vec model trained on Wikipedia data (from SIADS 655), then reduced to 30-dimensions using PCA. This is a strong data set because it captures both sentence-level features like length and average concreteness score, and word features via the word vectors. We also tested the effect of adding our unsupervised learning classes (topic classification for each sentence via LDA), but found that did not improve performance. Finally, while time is not an essential component of this data set, we believe it is helpful to have relatively current data which captures any recent trends or changes to how children acquire and learn language. To that end, the majority of the data is less than ten years old.

Methods and Evaluation

Model Pipeline

One goal of this project was to test many different features, models and hyperparameters. Because we knew this could be a time-consuming process we began by building a modeling pipeline that was modular and flexible Specifically, we added many checkpoints that saved our intermediate data transformations, allowing to repeat the smallest number of steps if/when we wanted to make a change to our preprocessing steps or feature creation. Most of the models we used were sklearn models, and all use the same API (fit, predict, etc.) so swapping those in and out was fairly easy. For our neural networks (Pytorch) and gradient boosting models (LightGBM) there was a bit more custom code, but we were still able to utilize the pipeline. Finally, we wrote small helper functions to make our workflow as efficient as possible. For example, we wrote a function that takes a fitted model and a "submission name" as arguments and creates predictions for the Kaggle test data set and automatically submits those predictions to the competition.

Models

As mentioned earlier, we tried a wide variety of models. Those models were:

- Custom “model” using only word length
- Random Dummy Model
- SGD Classifier
- Random Forest Classifier
- Fully-connected Neural Net
- Gradient Boosted Model

Performance Metric

We evaluated our model performances on a wide variety of metrics, but ultimately used accuracy as the decisive performance metric. We made this choice for several reasons. First, accuracy is what is used to score the Kaggle competition and it made sense to align. Second, with a balanced data set accuracy does not contain some of the pitfalls associated with the accuracy of a model on an unbalanced data set (i.e. obtaining a high accuracy score by always predicting the dominant class).

Third, there was no context provided for how this model would be used in downstream processes, which could have affected our choice. In a real world setting we would have worked with the business stakeholders to understand the relative costs of a false negative versus a false positive, which may have pushed us to use precision or recall. But with this information not available, and the combination of reasons above, we felt accuracy was the best metric to evaluate the performance of our model.

Final Model

In the end our final model choice was simple: a random forest. A random forest obtained the highest accuracy scores across all data sets, and is a fairly intuitive model, making it easy to explain to a non-technical audience. This balance of good performance and interpretability made a highly tuned random forest model the perfect choice for this project.

Results by Model Type

As mentioned above, we tested multiple models. Accuracy scores for all the models on the test data set, and the Kaggle datasets can be found in Table 2 in the appendix.

Model Evaluation

We evaluated our model using a wide variety of metrics and tools, looking at the performance, the importance of different features, and the importance of different hyperparameters.

To begin, we looked at how our model performed on a suite of traditional performance metrics and the confusion matrix.

Metrics

Metric	Score
Accuracy	70%
Precision	71%
Recall	70%
F-1	70%

Confusion Matrix

	Predicted 1	Predicted 0
True 1	36,534	15,468
True 0	15,119	37,071

Despite not using the ROC curve to compare models, it was still generated. Again, even though a precision-recall curve was not necessary to understand and balance different error types, it was still generated as well. These curves can be found in the appendix.

To understand the specifics of how our model was making the predictions it was making we started with traditional variable importance measures. The following visualization (from [Sklearn's documentation]) shows the average Gini importance (i.e. mean decrease impurity) of a feature across all trees in the random forest.

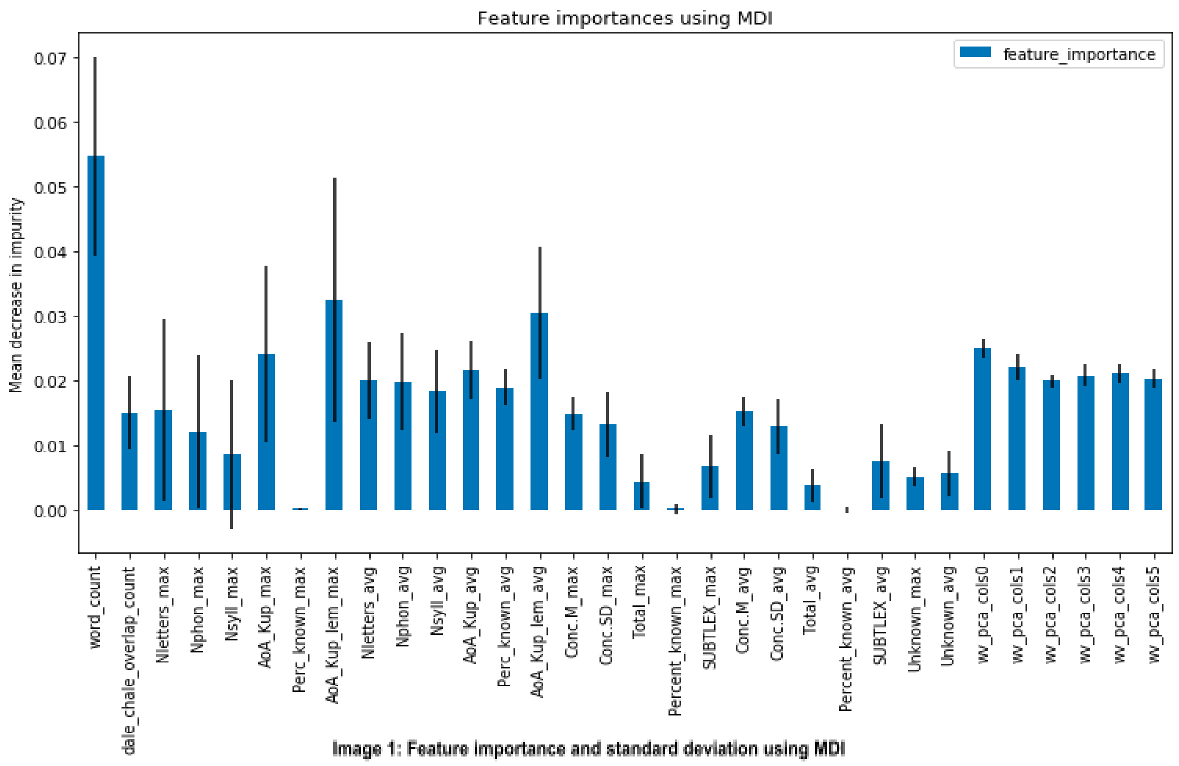


Image 1: Feature importance and standard deviation using MDI

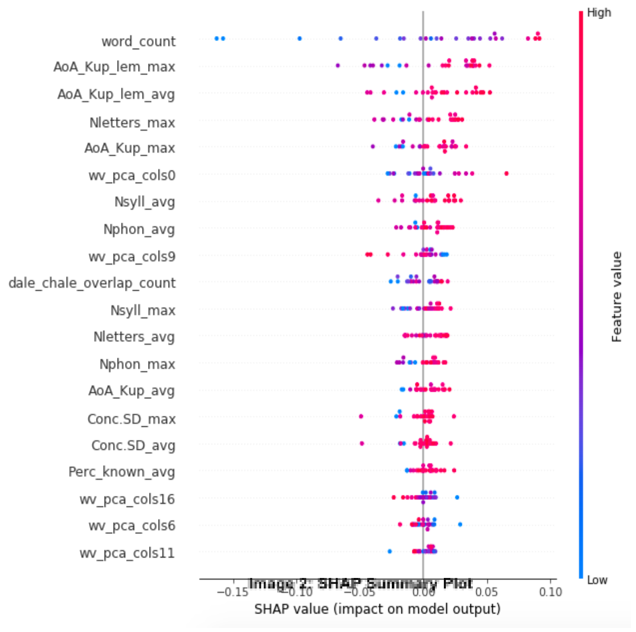


Image 2: SHAP Summary Plot

From this we can see that the word length is the most important feature (more on this later), followed by the max and average Age of Acquisition of the words in the sentence. After that all the word vector dimensions contribute roughly equally (trimmed for ease of viewing) and the rest of the features provide small amounts of value.

To expand on the basic feature importance metric, we used SHAP to understand not only which features are important, but how the values of those features affect the predictions made by the model.

Again we see many of the same features near the top of the chart, indicating their importance. Additionally, we see how the values of the features correlate to their effect on the final prediction. For example, we can see that shorter word count (blue dots) have negative SHAP values, meaning shorter sentence length variable value has a negative weight (pushes a prediction towards 0, or no simplification). This is not surprising given the distribution of the target variable by sentence length (Appendix Charts 1 and 2) and the fact that we built a “model” that was 61% accurate using sentence length as the only feature [2].

To further investigate the model features, we performed an ablation analysis, first training the model on only sentence features (1-28, Table 1), then on only word vector features (29-56, Table 1).

Dataset	Full	Sentence features only	Word vectors only
Accuracy	70.3%	70.1%	66.3%

The ablation analysis tells us that while both types of features are useful and can build a model that does much better than random guessing, the information contained in the features is redundant. The model including only sentence features has roughly the same accuracy as the model containing sentence features and word vectors. Likewise, the model trained on only word vectors performs slightly worse than the model trained on the full data set. In other words, adding word vector data to the model does not improve the accuracy; in a production environment these variables would be removed to reduce the amount of computation and complexity of the model.

<u>Hyperparameter Importances</u>

After testing many hyperparameter combinations, we are confident that we have found a very high-performing set of hyperparameters, and that there is a large set of hyperparameters that will provide similar performance. Across both our grid search and random search, we found most combinations only resulted in a small change in accuracy on the test set.

We also tried Optuna to optimize our hyperparameters, and while there was no boost in the model performance using the resulting hyperparameters, Optuna does provide some interesting analysis around hyperparameter choices.

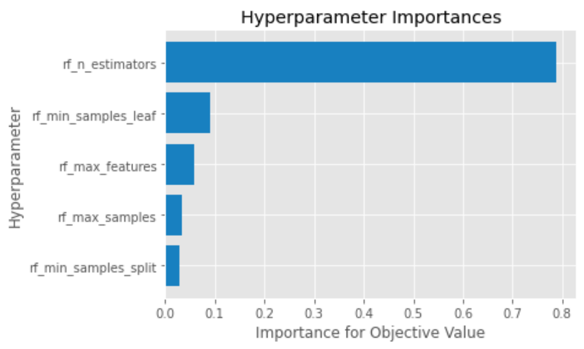


Image 3: Hyperparameter importance calculated via the FANOVA algorithm

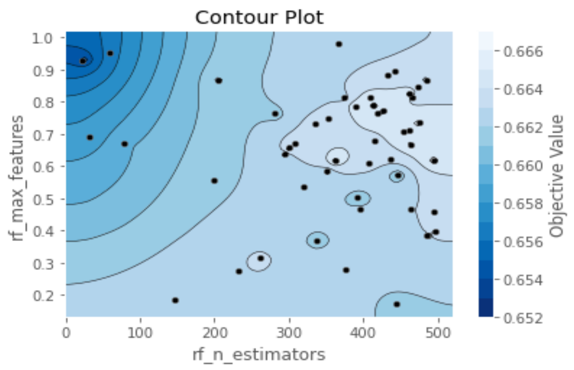


Image 4: Contour plot showing the relationship between n_estimators and max_features and their resulting accuracy.

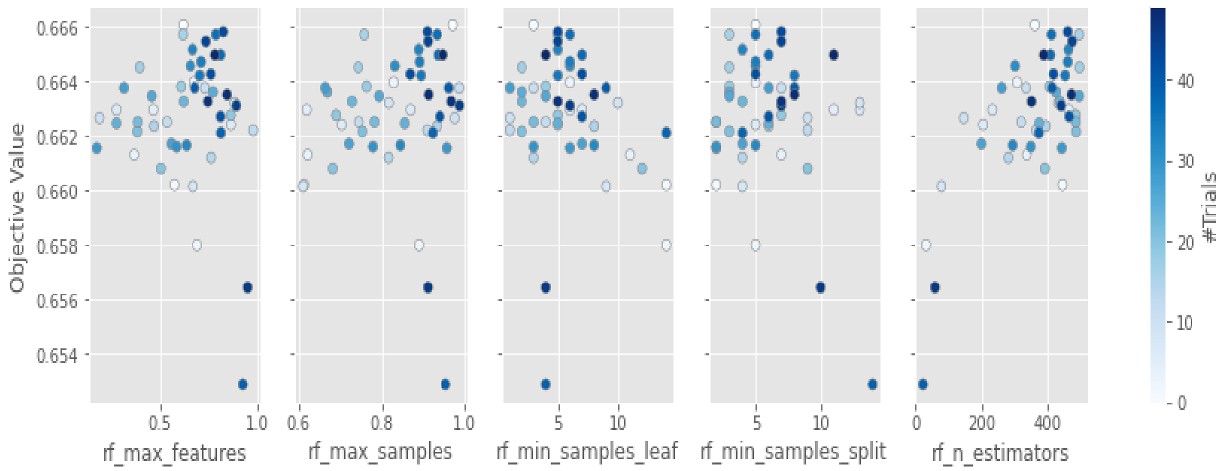


Image 5: Different hyperparameter values and their accuracy scores, color-coded by number of trials (darker dot = more tested parameter value)

Trade-Offs

This project was an exercise in balancing trade-offs – every decision had implications for the rest of our project. To highlight a few of the tradeoffs:

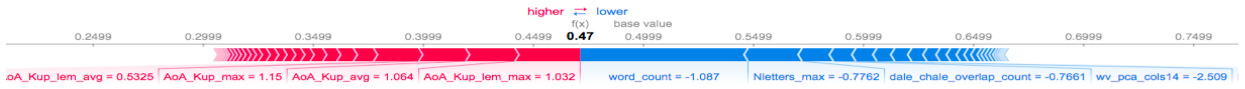
- *Model complexity versus explainability.* We spend most of our time building a highly accurate Random Forest (over gradient boosted model or neural network) because we wanted to ensure we built a model that would be easy to explain and interrogate. We were not interested in building a high-performing "black box" model where we would struggle to explain the predictions of our model.
- *Code readability and maintenance versus agility*. We wanted to move fast and try lots of things (features, models, etc.) However, we purposefully decided to move a little slower and write DRY, modular code. In the end this may have resulted in us trying a slightly smaller number of parameters or models, but it ensured everything we did try was reproducible with well-documented results.
- Training time and resources versus exploration. There are many types of models with infinite combinations of hyperparameters. Choosing how many models to explore and how in-depth hyperparameter searching was a constant tradeoff. Giving up hours and hours of watching Python test different hyperparameter combinations had to be balanced with quick, small changes that could easily be evaluated. In the end we felt we found a very good balance, allowing us to explore many models and options, but also getting quick, directed feedback that allowed us to explore in the right direction.

Failure Analysis

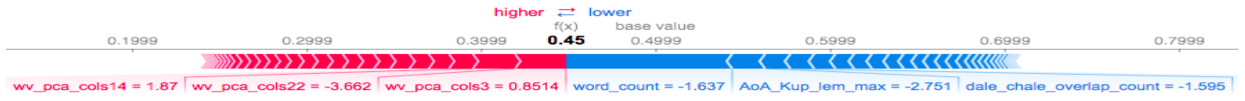
Given the importance of sentence length in our analysis, and the extreme class imbalance that exists for very short and very long sentences [Appendix Table 3 and Table 4], it is of no surprise that our model failed most often when a short sentence needed to be simplified or a long sentence did not.

The SHAP visualizations below each explain a single prediction. The first two examples show a positive class incorrectly identified as negative, the third example shows the opposite. In all three examples, the largest coefficient is associated with word length, which pushed the prediction below of the 0.5 threshold for the first two, and above it for the third. Being the most important feature, it is not surprising that a sentence with an actual class that does not match the predominant class for that word length is incorrectly classified.

Failure Ex. 1 - Actual Class: 1 Predicted: 0 (0.47)



Failure Ex. 2 - Actual Class: 1 Predicted: 0 (0.45)



Failure Ex. 3 - Actual Class: 0 Predicted: 0 (0.73)

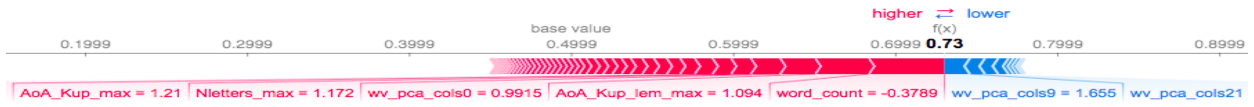


Image 6: SHAP plots visualizing single examples and how their feature values contributed to the predicted probability

While the above visualizations show only three failures, it exemplifies the pattern we saw over and over again when our model made an incorrect prediction.

Unsupervised Learning

Motivation

We were unsatisfied with the score from the first pass of our classification models, and we brainstormed new ways to expand our features list. We decided that we could add the results from an unsupervised learning model as an additional feature for our classification models. We did not know how, if at all, our classification models scores' would change by introducing a new feature. We ultimately decided on a **Latent Dirichlet Allocation (LDA)** model to perform topic modeling, and use the resulting topic of each sentence as a new feature for our classification models. We hoped that topic modeling would expose patterns in the difficulty of the text that was previously unseen.

Data Source

We used the same Kaggle data as described in the *Supervised Learning - Data Source* section. The only part needed from the data for an LDA model was the text from the main Kaggle data set, no additional data sources were necessary.

Unsupervised Learning Methods

Data Preprocessing

After loading the data, we created functions for preprocessing the text. In the initial pass, we converted all the text to lowercase, replaced hyphens in hyphenated words like second-most-populous with spaces, and trimmed any additional spaces. Next*, lemmatize,* and *stem* functions to apply to every token using NLTK's *word_tokenize* on the text. NLTK's *WordNetLemmatizer* is important in topic modeling, because it removes inflectional endings from words to capture the base meaning of the word. For example, the words *sang* and *singing* are converted to *sing.* For stemming, we used NLTK's *SnowballStemmer*, which is a more aggressive and improved version of the *PorterStemmer*. After stemming, we retain the root of every word and can continue with our topic modeling.

Bag-Of-Words
A bag-of-words (BoW) is an important concept in natural language processing, as it allows us to extract features from text that can be added to our models. The BoW model takes in text from documents and results in a vocabulary of known words with the total frequency of those words. Order and structure are discarded, leaving the model to infer if documents are similar when they share common words and frequencies. We decided to explore Gensim to make a Bow model and started with using the *corpora.dictionary* class to get the frequency of word counts in each document. We chose this over traditional ways of getting frequency counts (i.e., Python dictionary), so that we could use the *filter_extremes* method. The *filter_extremes* method filters out words that are very common or very rare. We chose to filter out words that appear less than 5 times in total and words that appear in more than 50% of documents.

TF-IDF Model
Initially, we tried to apply term frequency–inverse document frequency (TF-IDF) weights to the words in each sentence based on the values from the bag-of-words models. See here for more information on the TF-IDF weight calculation. However, we eschewed this idea after further research confirming that an LDA model only needs to accept a BoW vector. Since an LDA model is probabilistic, it estimates probability distributions for topics based on the words in the topics, for which TF-IDF is not necessary. Using TF-IDF may produce different results, but not necessarily better results.

LDA Model
We used Genism's *LdaMulticore* model which is generally faster than the *LdaModel, a*nd included the TFIDF transformed corpus, and dictionary as parameters. We additionally included the *workers* and *passes* parameters to speed up training time. *LdaMulticore* also takes *num_topics* as a parameter, which we determined to be 22 using the function described in the *Number of Topics Optimization* section. The main drawback of using *LdaMulticore* is that the probabilities returned after each run are not reproducible, even with a random state set. This was not discovered until our analysis was complete, and we decided not to revisit the model. In hindsight, we should have used the *LdaModel*. The idea for using an LDA model came from this lecture by Carlos Lara.

Add Topic Number as Feature
The function *extract_topic* simply extracts the topic number with the highest probability for a given sentence, and adds it back to the original *WikiLarge_Train* dataset and then saved as a csv. The topic numbers are then included as features in the supervised learning model.

Unsupervised Evaluation

Coherence Score
Coherence score is an evaluation method for LDA models that measure how interpretable topics are to humans. We used Gensim's *CoherenceModel* with the *coherence = 'c_v'* parameter to calculate the coherence score of our model. C_V Coherence is a commonly chosen and default metric to calculate coherence score. The coherence score from using 22 topics was 0.475. There is nothing that truly determines whether a coherence score is good or bad, as it depends on the data collected. We ultimately concluded that a coherence score of 0.475 was acceptable due to the size and noise of the text data. We discovered after we trained our model and used the results in the supervised model, that the authors of the C_V calculation recommend no longer using it, as described here. However, we were already deep into the project, and determined that the potential negative impacts were not severe enough to warrant running our entire pipeline again.

Perplexity Score
Perplexity is a statistical measure of how well the model predicts a sample. A model with the lowest perplexity can normally be considered the best. We originally understood the *lda.log_perplexity()* function to calculate the full perplexity score. However, we kept getting negative numbers. We were close to scrapping the perplexity score all together, until we discovered that *lda.log_perplexity()* actually returns a likelihood bound, which can be passed into *numpy.exp2(bound)* to calculate the actual perplexity score as described in this post.

Number of Topics Optimization

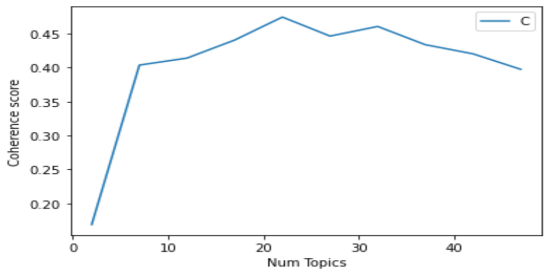


Image 7: Coherence Score per *n* topics

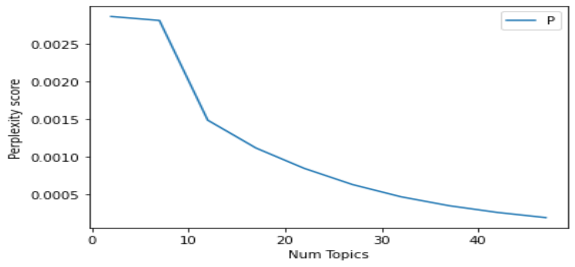


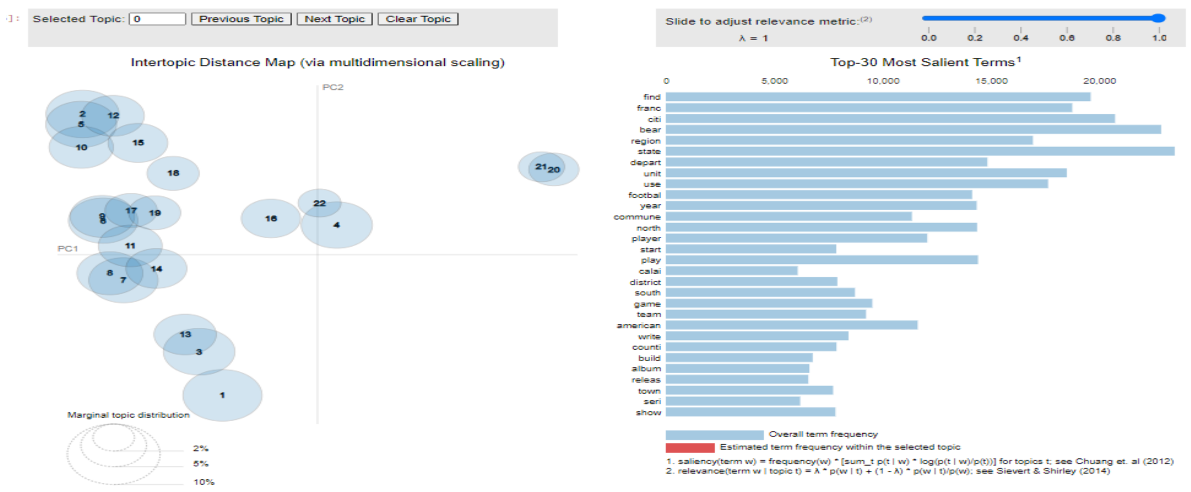
Image 8: Perplexity Score per *n* topics

After our initial attempts, we realized that fine-tuning the *num_topics* parameter would give us higher performance metrics, specifically *coherence_score*, to further refine our topic model. To do this, we iterated through 5 topics at a time, starting with a minimum of 2 topics, and collected the coherence scores for each model as seen in the figure to the left. The coherence score peaked when number of topics = 22.

We also used perplexity scores to determine the optimal number of topics, but given our uneasiness about the perplexity score calculation described above, we stuck with coherence score as the main determinant. Perplexity score did start to approach a lower number around the same number of topics as the highest coherence score.

LDAvis

LDAvis is a great tool to visualize LDA models. It computes summary statistics and creates an interactive visualization where you can select the topic and view the words that make up the topics. The interactive version can be viewed in the notebook.



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)

Image 9: LDAvis Example

Discussion

Supervised Learning Discussion

In the supervised learning part, we learned about the libraries available to tune hyperparameters (Optuna) and evaluate trained models (SHAP). We also learned the value of building a pipeline, and that most ML projects are fairly similar: data -> features -> train model -> make predictions. What surprised us the most was the similar performance across many model types and hyperparameter combinations. The power of complex neural networks and sophisticated gradient boosting models are often touted, but they did not perform as well as a more traditional technique: random forest. This led us to speculate that there was a fair amount of noise in the data: no matter how complex a model was used, or what features were derived, there was an upper bound on accuracy due to the imperfect labeling of the sentences. There were sentences that had almost the exact same structure and characteristics as each other, one of which needed to be simplified and the other which did not. Our features were strong, and our pipeline was stable, any additional time would have been best used cleaning up the data and ensuring that correct, sensical labels were created (and perhaps trimming the data set to only include sentences of more than X words).

The most obvious ethical issue attached to this project concerns the data labels and how they were generated. Was the training data classified by a particular group of people with certain biases about what needs to be simplified and what does not? Are there sentences that might not appear to require simplification, but are an artifact of the classifier's opinion rather than some ground truth. This could have many downstream implications such as certain groups reading the simplified version of a text but it still not being simplified enough for them to comprehend. To generalize, if there is bias in the model or the data, deploying a machine learning task only encodes these biases. If there are decisions resulting from the use of the model, maybe deciding if an ESL student can graduate to the next level or a reading assessment, the model and data should be rigorously reviewed for fairness and regularly retrained and updated. While there is still some chance biases may be captured in the model, or via the data, critically reviewing the model and the uses of its outputs can go a long way in preventing misuse.

Unsupervised Learning Discussion

With unsupervised learning, we learned a great deal about how to effectively use and evaluate an LDA model. Originally, we believed that using TF-IDF weights was necessary to train an LDA model, but discovered that we could use the BoW dictionary to the same effect. Using 22 topics resulted in interesting, but not unexpected results. Many topics were understandable, but some topics did not make a whole lot of sense from a human perspective. One topic had base words like "univers", "school", "studi", "college" in it, which made it easy to infer the topic, however, another topic had "japan", "class", "guitar", "discov", and "angel", and the topic was not clear. The model found its own patterns in the text, which could be good or bad. In order to find out, we tested the initial results in our classification model, but we did not see any change in score. With that, we concluded it wasn't worth the time or effort to continue tuning the parameters for a marginal increase in score. We could have added *gamma_threshold*, which would have likely improved coherence and perplexity scores, but we felt that we gained as much insight as we needed from the unsupervised learning model. Additionally, we could have added the text from the test data to further expand our model. Unfortunately, the pipeline of training a new LDA model, calculating max scores, and then feeding the topics into the supervised model became too cumbersome. It would have been interesting to see if some combination of parameters would improve our classification score.

A major ethical concern with using topic modeling in the way that we used (as a feature in a classification model), ties into the supervised learning ethical concerns described above. The topics could find patterns that really have nothing to do with simplifying sentences, and lead to the classifier making mistakes. Topic modeling on its own does not really lead to any ethical concerns, however, the results from topic modeling can be used in recommender systems, sentiment analysis, and text summarization models. If the topic modeling is incorrect or undertuned or overtuned, it may fail in providing the adequate information to these other models. A failed text summarisation system could result in interested parties receiving the incorrect information and lead to the spread of misinformation. This could have the same effect on a recommender system. Ultimately, we do not believe there are major ethical concerns with using topic modeling in exploratory analysis, but it is important to be aware of the final goal of using the topic modeling results.

Statement of Work

The two of us worked closely on strategizing ideas for supervised and unsupervised learning, and what goals we ultimately wanted to achieve. John took the lead on the supervised learning section, as his professional experience was essential in making choices and selecting the right parameters. John wound up writing the entire supervised learning section of this paper. Holden focused on brainstorming how we could use unsupervised learning to improve or supervised learning results, did research on topic modeling, and wrote the unsupervised learning section of this paper. Both group members collaborated together to make sure our results worked together.

Appendix

Table 1: Variable Descriptions

	Variable Name	Description	Source Document	Data Type
1	word_count	Number of words (including punctuation) in sentence	WikiLarge_Train.csv	Numeric
2	dale_chall_overlap_count	Count of Dale Chall words in sentence	dale_chall.txt	Numeric
3	aoa_overlap	Count of Age of Acquisition words in sentence	AoA_51715_words.csv	Numeric
4	Nletters_max	Max length of word in sentence	AoA_51715_words.csv	Numeric
5	Nphon_max	Max phoneme length of word in sentence	AoA_51715_words.csv	Numeric
6	Nsyll_max	Max syllable length of word in sentence	AoA_51715_words.csv	Numeric
7	AoA_Kup_max	Max AoA age of word in sentence	AoA_51715_words.csv	Numeric
8	Perc_known_max	Max percent of people who know a word in sentence	AoA_51715_words.csv	Numeric
9	AoA_Kup_lem_max	Estimated max AoA	AoA_51715_words.csv	Numeric
10	Nletters_avg	Avg. length of word in sentence	AoA_51715_words.csv	Numeric
11	Nphon_avg	Avg. phoneme length of word in sentence	AoA_51715_words.csv	Numeric
12	Nsyll_avg	Avg. syllable length of word in sentence	AoA_51715_words.csv	Numeric
13	AoA_Kup_avg	Avg. AoA age of word in sentence	AoA_51715_words.csv	Numeric
14	Perc_known_avg	Avg. percent of people who know a word in sentence	AoA_51715_words.csv	Numeric
15	AoA_Kup_lem_avg	Estimated avg. AoA	AoA_51715_words.csv	Numeric
16	Conc.M_max	Max. concreteness of word in sentence	Concreteness_ratings_Brysbaert_et_al_BRM.txt	Numeric
17	Conc.SD_max	Max. Standard deviation concreteness of word in sentence	Concreteness_ratings_Brysbaert_et_al_BRM.txt	Numeric
18	Total_max	Max. word number of people surveyed	Concreteness_ratings_Brysbaert_et_al_BRM.txt	Numeric

19	Percent_known_max	Max. word number of people who knew word	Concreteness_ratings_Brysbaert_et_al_BRM.txt	Numeric
20	SUBTLEX_max	Max subtlex score of word in sentence	Concreteness_ratings_Brysbaert_et_al_BRM.txt	Numeric
21	Conc.M_avg	Avg. concreteness of word in sentence	Concreteness_ratings_Brysbaert_et_al_BRM.txt	Numeric
22	Conc.SD_avg	Avg. Standard deviation concreteness of word in sentence	Concreteness_ratings_Brysbaert_et_al_BRM.txt	Numeric
23	Total_avg	Avg. word number of people surveyed	Concreteness_ratings_Brysbaert_et_al_BRM.txt	Numeric
24	Percent_known_avg	Avg. word number of people who knew word	Concreteness_ratings_Brysbaert_et_al_BRM.txt	Numeric
25	SUBTLEX_avg	Avg. subtlex score of word in sentence	Concreteness_ratings_BrysbaConcreteness_ratings_Brysbaert_et_al_BRM.txtert_et_al_BRM.txt	Numeric
26	UNKNOWN_max	Max. word number of people who did not know word	Concreteness_ratings_Brysbaert_et_al_BRM.txt	Numeric
27	UNKNOWN_avg	Avg. word number of people who did not know word	Concreteness_ratings_Brysbaert_et_al_BRM.txt	Numeric
28	wv_pca_cols[0-29]	Avg Word Vector	wikipedia.100.word-vecs.kv.vectors.npy	Numeric

Table 2: Model Performances Across Datasets

Model Type	Test Accuracy	Kaggle Public Score	Kaggle Private Score
Dummy Classifier	49.9%	48.9%	49.3%
SGD Classifier	63.3%	72.35	72.9%
Random Forest (default hyperparameters)	70.7%	72.4%	72.8%
Random Forest (tuned hyperparameters	71.1%	73.8%	74.3%
LightGBM (default hyperparameters)	67.4%	68.5%	68.4%
LightGBM (tuned hyperparameters)	68.3%	68.1%	68.6%
FC Neural Net	68.5%	69.7%	69.8%

Chart 1: ROC Curve: Visualization of the tradeoff between true positive and false positives

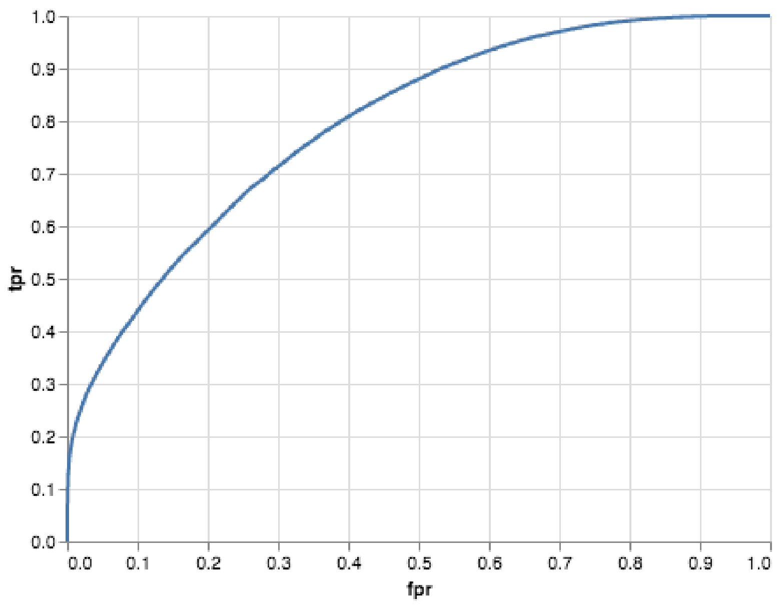


Chart 2: Precision Recall Curve: Visualization of the tradeoff between precision and recall

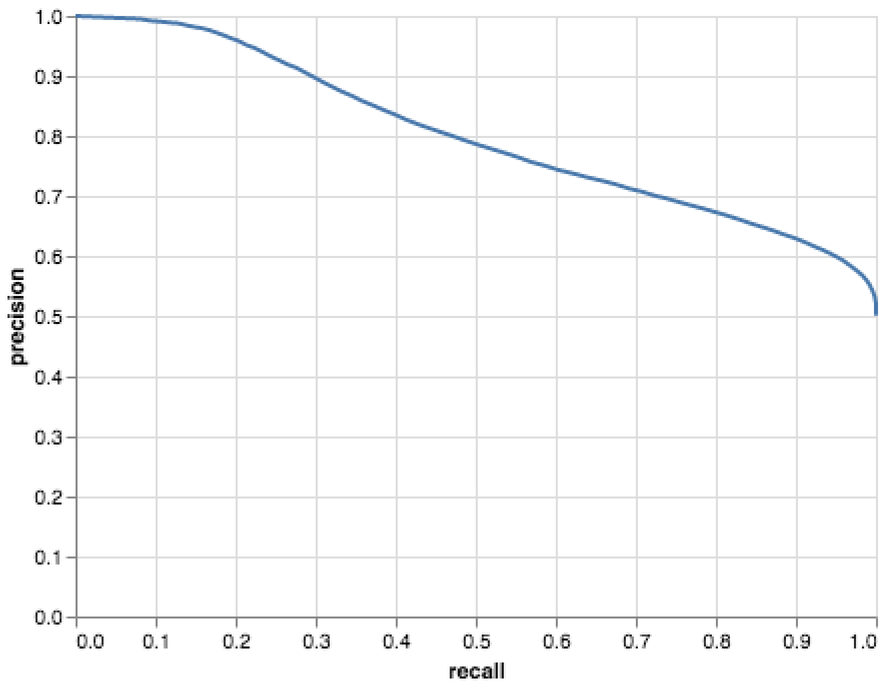


Chart 3: Percent of Sentences to be Simplified (target variable = 1) by Word Count

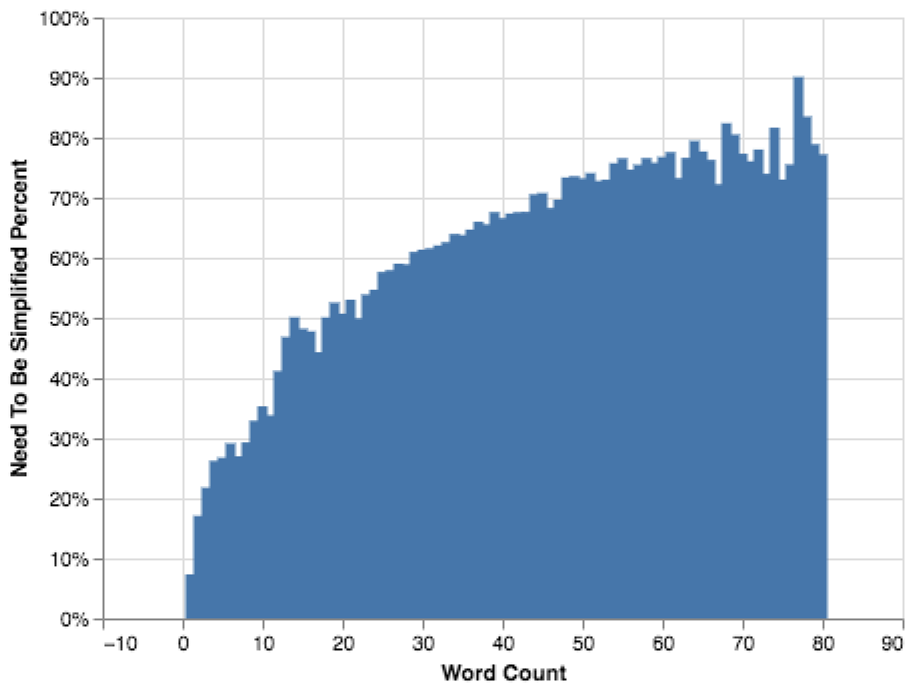
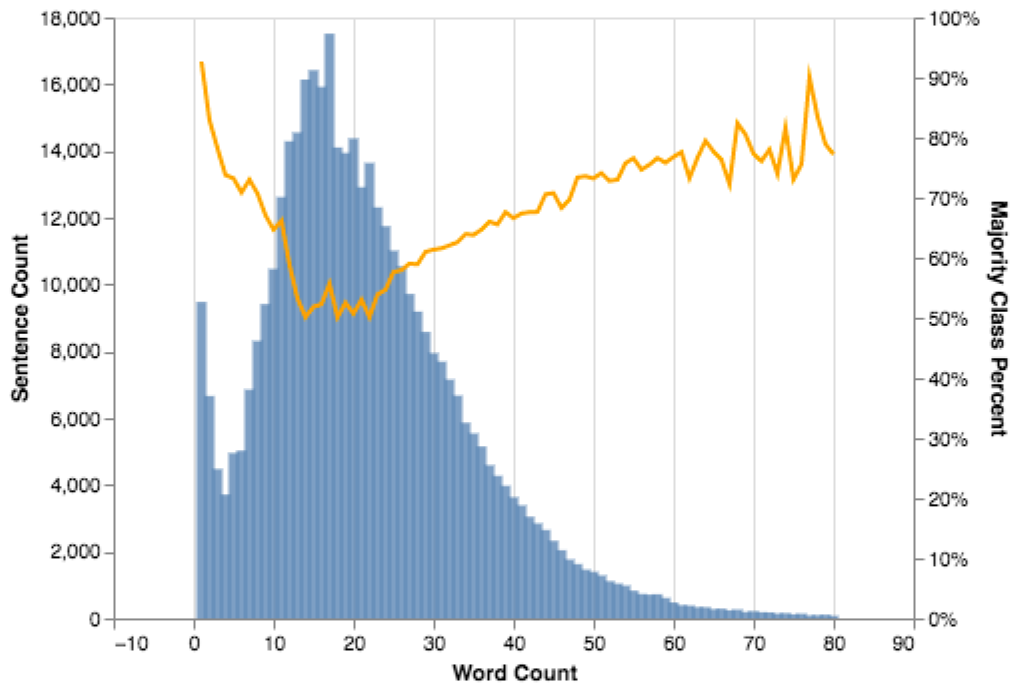


Chart 4: Majority Class Probability by Word Count



Notes

[1] kaggle competitions download -c umich-siads-695-fall21-predicting-text-difficulty

[2]

```
lower_bound = 0.45
upper_bound = 0.55
correct_count = 0

# for every sentence, predict the majority class based on sentence length
# if the ratio of need to be simplified : does not need to be simplified
# is greater than the lower_bound and less than the upper_bound
# make a random guess

for idx, row in X_test.iterrows():
    current_word_len = row.word_count
    word_1_prob = model_lookup_df.loc[model_lookup_df['word_count'] == current_word_len, 'label'].values[0]
    if word_1_prob < lower_bound or word_1_prob > upper_bound:
        prediction = int(np.round(word_1_prob))
    else:
        prediction = random.choice([0,1])
    if prediction == row.label:
        correct_count += 1

print(f'{correct_count} of {len(X_test)} guesses correct - {(correct_count / len(X_test)*100):.2f}% correct')
# 60.2%
```