Milestone II Project Report

Tyler Allerton, James Beck, Kevin Borah

Introduction

Our team chose to use natural language processing methods on data drawn from Wikipedia. The task was to train a model using supervised and unsupervised techniques to classify whether a given sentence needs simplification for impaired learners or if it is already in simplified form. In our ever-increasing inclusive and globalized world, it is essential that textual information be accessible and comprehensible to readers spanning the spectrum of reading proficiencies. The data was provided as the default option by the instruction team and was downloaded from the Kaggle website.

The data consisted of three comma-separated values documents: WikiLarge_Train.csv (the training set), WikiLarge_Test.csv (the test set without labels) and sampleSubmission.csv (a sample submission file in the correct format). The sample submission file was used to submit the outputs from the model to the Kaggle site for accuracy scores. The accuracy scores provided through Kaggle only represented the accuracy for 50% of the data. At the end of the experiment the accuracy scores were to be revealed. The training data consisted of a test variable drawn from Wikipedia and a label of whether the text is or is not in simplified form. In total there were 416,768 rows of data in the training data. The test data followed the same textual format but did not include the labels. The test data consisted of 119,092 documents. There was no time component to any dataset included in the project.

Our project also made use of the following data sources included in the Kaggle data package:

- The Dale-Chall list of elementary English words that are familiar to 80% of American 4th grade students.
- A list of Concreteness Ratings which evaluate the degree to which the concept denoted by a word refers to a visual entity. It is believed that concrete words are easier to remember than abstract words because they activate perceptual memory codes in addition to verbal codes.
- A list of Age of Acquisition words which approximate the age in which a word was learned. It is understood that this word list will be of limited value due to the difference between child and adult language learning. However, it could be used as a differentiator when words are evenly used in simplified and difficult texts.

Our team was motivated to take on this challenge and specific data set because we were all collectively intrigued by previous course material related to natural language processing and wanted to further our understanding and skill sets within this area. Out of the courses in the applied data science curriculum, NLP was one of the most fascinating subjects and a offered only

a glimpse into its potential in real-world applications. We also chose to look at NLP because it was our weakest subject. It wasn't weak because of the course content; instead, it was weak because of our previous exposure to the subject. We all have varying degrees of familiarity with statistical methods involving numerical variables but none with text values. So, in essence, this project was more about filling in the gaps of what we didn't learn than it was about demonstrating what we did learn.

Another motivation that is closely related to our first is the ability to compare our efforts with those of others in the class. We like the idea of learning from doing, but we can also appreciate the concept of learning from comparison. It was our hope that many of the teams would choose to use the default dataset. Our thought was to be able to share code with other teams after the final report deadline. The sharing might lead us to learn more about the inner workings of creating a thorough NLTK model and how to communicate those findings through meaningful visualizations.

Supervised Learning

Supervised Learning Methods

There are many methods that can be applied to Supervised Learning, and we chose many of them. The beauty of today's computing environment is multiple methods can be applied to a single data source, the models are computationally inexpensive (at least in the case of this project), an exhaustive search over parameter values can be applied to each method, and the methods can be compared to select the best option for that data. In total, we ran more than 200 models in the supervised learning section alone. We chose to run a Dummy Classifier for our baseline, and then compared that baseline against results from Naïve Bayes, Decision Trees, Random Forest, Linear Support Vector Classifier, and finally a Logistic Regression. We provided the results of each option below in order of accuracy.

Train, Validation, and Test Data

The results are reported in terms of accuracy based on the guidelines of the accompanying instructions. The accuracies reported in most cases will refer to the validation accuracy. We chose to split our training data into a train and validation set. We used that validation set to perform two functions. First the validation set helped us evaluate the models against one another so that we could select the best model for our training data. Secondly, it helped us to optimize the model parameters. The accuracy for the actual test data will be presented in the final section.

Count and Term Frequency-Inverse Document Frequency Vectorizers

We used the Count and Tf-Idf vectorizers for every method mentioned below. Tf-idf vectorizers numerically represent documents by measuring term frequency and inverse document frequency.

Term frequency gives weight to words which appear more often, and inverse document frequency applies more weight to words which appear in fewer documents. Count vectorizers represent textual data numerically as well, but only capture the count of words per document. The difference between their implementation was negligible. We will not comment on the accuracy of each because of the impracticality. We will highlight that for each vectorizer we omitted the max_feature parameter. We also set the max_feature parameter to 13,601, which is 10% of the total number of features in our training set. We settled on this number after conducting several runs of the Logistic Regression and Linear Support Vector Classifier with varying percentages of the data. 10% proved to be the optimal percentage, though we didn't investigate fractional values that weren't factors of 10. We will discuss this in more detail below. In the meantime, we will highlight the specific vectorizer that produced the best result at the end of each paragraph and the percentage of the features.

Data Standardization

We explored various standardization methods for preprocessing the data. The scikit-learn library offers several methods, but based on our data type, we chose to focus on two main methods. We used the Standard Scaler which normalizes the data to a zero mean and unit variance. We then looked at the Max Absolute Value Scaler which scalers each feature individually so that the maximum absolute value of each feature in the training set equals 1 and does not destroy the sparsity of the data. These two standardization methods were used with the Logistic Regression and Linear Support Vector Classifier methods. The results of the models were not enhanced by the scaling, so we chose to omit this step from other classification methods.

Random Guessing

As with any good modeling project, we needed a baseline to compare against more complex classifiers. Scikit's Dummy Classifier was a perfect solution. We used the uniform strategy to generate uniformly random predictions based on the counts of the y parameter. The accuracy of the Dummy Classifier was 0.500 for the training set and 0.502 for the validation set. We chose not to run this method against the test set because it produced the worst results out of all the methods we chose. We did expect the results from this model to be 50/50; not because it was a random choice, but because the y parameter itself was split 50/50 between 0 and 1.

Naïve Bayes

Naïve Bayes models are fast to train and work well with high-dimensional sparse data (Guido, 2017). Because of their quickness, they are used in lieu of linear models that may take too long when using large datasets (Guido, 2017). The scikit-learn reference guide even highlights the model's utility in document classification and spam filtering (Scikit, 2022). We explored the utility of the Gaussian, Multinomial, and Complement Naïve Bayes methods. The Gaussian method failed because it required a dense matrix rather than the sparse matrix we provided. The

error message did provide an option for converting the inputs to a NumPy array. We converted the training and validation data to an array, tried to fit the model, and then tested the accuracy. However, this proved to be extremely difficult in terms of hardware and computational resources; the computer quickly ran through 128GB of RAM in less than a minute. Instead of wasting time on a method that was clearly not intended for sparse data, we moved on to the other two methods. With the full feature set, the model did not generalize well for the Complement or the Multinomial. The accuracy for the training data was 0.724 and the validation accuracy was approximately 0.595 for both methods, indicating an overfitting of the models. With the vectorizer max_feature parameter set to 10% of the training features, the performance of these methods improved, producing an accuracy of 67% for the training data and 64% for the validation data. The difference between the training and validation accuracy did not lead us to believe the model was underfitting or overfitting. We did adjust the alpha parameter for values between 0.1 and 1, in 0.1 increments, for the sake of exploration. The alpha parameter when set to 1 is called Laplace smoothing and Lidstone smoothing for any value less than 1. The best accuracy produced for the validation set for either method was 0.64709. The parameters used were an alpha value of 1, max_features were set to 10% in the Count Vectorizer, and the implementation of the Complement method. In order to distinguish the best outcome, we had to check to the fifth decimal place.

Decision Trees

Decision Trees are a well-recognized method for classification tasks and typically have the advantage of being both simple to understand and interpret (Guido, 2017). We say *typically* because models with more than 100,000 features, like ours, are difficult to understand and visualize no matter which technique is applied. One of the disadvantages of decision trees is that they tend to over generalize the data and work well in training, but not so well in testing and production (Guido, 2017). As it relates to our project, this disadvantage proved true and presented itself in this exact way. The decision tree produced the highest testing accuracy of any model at 0.986. The closest value for testing accuracy in any other model was 82% for the logistic regression and linear support vector classifier methods. The validation accuracy was only 0.671 which, as we alluded to earlier, indicates over generalization. Pruning is a common strategy to prevent overfitting. The depth of our decision tree was 3,828. We chose to use the max_depth parameter as our initial pruning method. Like our max_features parameter used in the vectorizers, we chose to investigate the max_depth parameter in 10% increments. Unfortunately, the max_depth parameter had little effect on the gap between the training and validation accuracies. Each of the depths tested resulted in training accuracies of 97% and validation accuracies of 66%. Ultimately, the best validation accuracy for this model was 0.670, used a max depth of 383, consisted of a full feature representation, and utilized the tf-idf vectorizer.

Random Forest Classifier

Random Forest Classification is a method for addressing the decision tree dilemma of overfitting (Guido, 2017). The random forest is a series of decision trees each with the same traits of overfitting but with decent accuracy when applied to the validation or test sets (Guido, 2017). However, the overfitting can be reduced by averaging the results across all decision trees in the random forest (Guido, 2017). In our case, we used the default value of 100 trees. It then averages the results for each of those trees to produce the final classification. We did not spend a significant portion of time exploring this method as this was a time-consuming process that returned similar values to the Decision Tree. The accuracy for the training data was 0.986 and the accuracy for the validation data was 0.687. Ideally, we wanted to use significantly more trees (if given the time), but for the sake of efficiency, we chose not to.

It is interesting to note that the Random Forest did not differ considerably from the Decision Tree in classification accuracy. However, the Random Forest Classifier performed much better with the reduced number of features in the vectorizer. The best validation accuracy was 0.690 and utilized the tf-idf vectorizer with a max_features parameter of 10%.

Linear Classifiers

The final two methods are two well-known and widely used linear classification algorithms, Linear Support Vector Classifier (LinearSVC) and Logistic Regression (Guido, 2017). Both methods worked well out of the box and produced training and validation accuracies that led us to believe these were the top choices for supervised learning. They were much faster to train than the decision trees and random forest but couldn't touch the speediness of the Naïve Bayes method. However, both models were exceptional choices with respect to speed and accuracy.

We used the C parameter as the main parameter for tuning the outcomes. The C parameter controls the L2 regularization (Guido, 2017). High values of C apply less regularization and try to fit as closely to each point as possible (overfitting). Small C values produce simple models and try to fit generally to the training data points (Guido, 2017). We tested several C values and Tables 1 through 4 show those C values in the column headers. The values in green indicate the C parameters with the highest accuracy score per feature set. A C parameter of 0.2 was the most accurate in every case for the LinearSVC fit (Table1) with the validation set, regardless of what percent of the total features were trained on the vectorizer. In Table 2, the most accurate C parameter shifts to higher values. This is to be expected because Table 2 represents the training data. That means as the C value increases so does the overfitting. This is clear when we start to compare the accuracy scores between the training and validation sets. For example, there is almost a 20-percentage point difference between the two scores when the C parameter is set to 80 and the vectorizer is fit with a full feature set.

Linear Support Vector Classifier Results

% of total features	C Parameters								
	0.2	0.4	0.6	0.8	1	20	40	60	80
10%	69.41%	69.40%	69.38%	69.36%	69.33%	69.26%	69.26%	69.27%	69.27%
20%	69.07%	68.82%	68.78%	68.69%	68.62%	68.33%	68.33%	68.33%	68.33%
30%	68.85%	68.52%	68.35%	68.21%	68.10%	67.35%	67.30%	67.28%	67.27%
40%	68.63%	68.35%	68.18%	67.96%	67.77%	66.66%	66.57%	66.52%	66.51%
50%	68.58%	68.13%	67.87%	67.58%	67.37%	65.82%	65.72%	65.65%	65.63%
60%	68.43%	67.99%	67.73%	67.43%	67.15%	65.47%	65.32%	65.23%	65.17%
70%	68.41%	67.89%	67.52%	67.26%	66.99%	65.16%	64.95%	64.84%	64.81%
80%	68.32%	67.72%	67.32%	66.99%	66.69%	64.60%	64.34%	64.27%	64.20%
90%	68.18%	67.51%	67.08%	66.72%	66.45%	64.03%	63.81%	63.69%	63.67%
100%	68.01%	67.34%	66.87%	66.50%	66.14%	63.53%	63.28%	63.18%	63.14%

Table 1: Linear Support Vector Classifier Accuracy Scores for Validation Set

% of total features	C Parameters								
	0.2	0.4	0.6	0.8	1	20	40	60	80
10%	72.09%	72.24%	72.27%	72.27%	72.28%	72.30%	72.31%	72.31%	72.31%
20%	73.45%	73.80%	73.95%	74.03%	74.08%	74.13%	74.12%	74.12%	74.12%
30%	74.23%	74.88%	75.19%	75.36%	75.44%	75.75%	75.75%	75.74%	75.74%
40%	74.69%	75.63%	76.06%	76.29%	76.45%	77.14%	77.16%	77.16%	77.17%
50%	75.02%	76.11%	76.67%	77.01%	77.22%	78.47%	78.51%	78.53%	78.54%
60%	75.24%	76.47%	77.10%	77.52%	77.79%	79.56%	79.64%	79.68%	79.70%
70%	75.40%	76.76%	77.48%	77.95%	78.25%	80.51%	80.65%	80.70%	80.72%
80%	75.55%	77.04%	77.86%	78.37%	78.74%	81.38%	81.52%	81.58%	81.62%
90%	75.68%	77.29%	78.21%	78.81%	79.21%	82.11%	82.27%	82.33%	82.35%
100%	75.80%	77.54%	78.53%	79.19%	79.63%	82.74%	82.89%	82.95%	82.99%

Table 2: Linear Support Vector Classifier Accuracy Scores for Training Set

The first column on the left-hand side of Table 1 and Table 2 represents the maximum number of features fit for the tf-idf vectorizer. The number of features was an important parameter in the choice of models. The results we uncovered in this exploration made us return and test the reduced feature space on every model used in our supervised learning. The reduced feature set was the best option for all models except for the Decision Tree. That makes sense as the Decision Tree is a model that thrives on overfitting and more features feed that mechanism. Fitting a model with 10% of the total features produced the best accuracy scores for both linear models. The cells marked in red indicate the best performing feature percentages for each model. 10% of the total features was always an optimal solution for the validation accuracy. 100% of the total features was the optimal solution for the training accuracy in every case except one in the Logistic Regression model with a C parameter of 0.2. In that single case, 90% of the total features was the best solution.

Logistic Regression Classifier Results

% of total features	C Parameters								
	0.2	**0.4**	**0.6**	**0.8**	**1**	**20**	**40**	**60**	**80**
10%	68.99%	69.26%	69.38%	69.44%	69.45%	69.26%	69.26%	69.27%	69.28%
20%	68.82%	69.07%	69.15%	69.14%	69.19%	68.64%	68.54%	68.49%	68.46%
30%	68.61%	68.93%	69.02%	69.02%	69.04%	67.96%	67.71%	67.61%	67.55%
40%	68.56%	68.86%	68.95%	68.93%	68.91%	67.53%	67.21%	67.10%	67.02%
50%	68.50%	68.83%	68.93%	68.91%	68.85%	67.07%	66.60%	66.40%	66.28%
60%	68.47%	68.75%	68.85%	68.81%	68.76%	66.76%	66.29%	66.07%	65.90%
70%	68.48%	68.79%	68.81%	68.82%	68.74%	66.60%	66.13%	65.83%	65.64%
80%	68.46%	68.75%	68.76%	68.74%	68.67%	66.21%	65.59%	65.28%	65.06%
90%	68.47%	68.74%	68.71%	68.68%	68.55%	65.83%	65.16%	64.72%	64.47%
100%	68.40%	68.65%	68.63%	68.60%	68.44%	65.50%	64.71%	64.28%	63.93%

Table 3: Logistic Regression Accuracy Scores for Validation Set

% of total features	C Parameters								
	0.2	0.4	0.6	0.8	1	20	40	60	80
10%	70.51%	71.27%	71.60%	71.80%	71.90%	72.33%	72.34%	72.35%	72.36%
20%	70.72%	71.74%	72.30%	72.64%	72.91%	74.25%	74.26%	74.27%	74.27%
30%	70.78%	71.97%	72.65%	73.05%	73.42%	75.77%	75.84%	75.84%	75.87%
40%	70.81%	72.06%	72.83%	73.30%	73.67%	76.95%	77.13%	77.21%	77.26%
50%	70.82%	72.12%	72.93%	73.44%	73.83%	77.89%	78.32%	78.47%	78.55%
60%	70.82%	72.14%	72.99%	73.54%	73.96%	78.65%	79.18%	79.39%	79.50%
70%	70.82%	72.17%	73.01%	73.59%	74.02%	79.27%	79.93%	80.25%	80.41%
80%	70.82%	72.18%	73.05%	73.64%	74.11%	79.91%	80.67%	81.01%	81.22%
90%	70.85%	72.21%	73.08%	73.72%	74.20%	80.52%	81.34%	81.73%	81.96%
100%	70.83%	72.23%	73.11%	73.77%	74.28%	81.02%	81.92%	82.34%	82.56%

Table 4: Logistic Regression Accuracy Scores for Training Set

The Logistic Regression followed a lot of the same trends and produced practically the same results as the LinearSVC. The higher C parameters produced higher accuracy scores for the training data. However, the Logistic Regression model trended toward the middle value of 1 for the C parameter while the percentage of total features was low. As the percentage increased, the values started drifting downward toward 0.4.

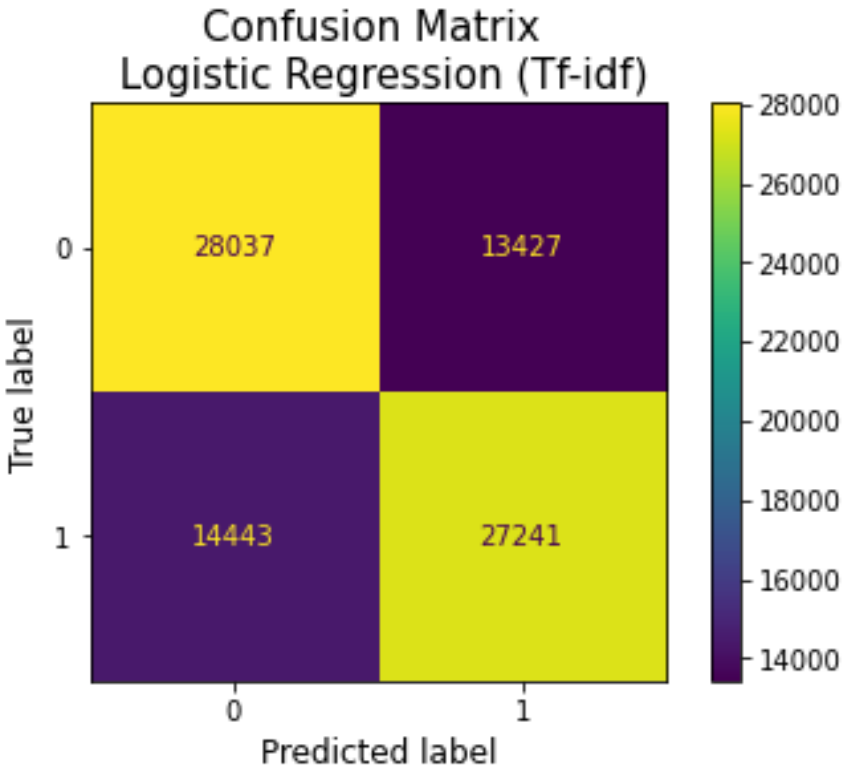
The final parameters we chose to explore was setting the number of n-grams. We tested the logistic regression model with n-grams set at 2 to 10. Due to the Logistic Regression being the optimal model prior to exploring the n-gram parameter we chose not to include the n-gram parameter in any LinearSVC run. In the end, the n-gram did help the Logistic Regression model and produced the highest accuracy score overall, 70.38%. The Logistic Regression ran most optimally with a C parameter of 1, n-gram set to (1, 2), and with 10% of the total features trained in the tf-idf vectorizer.

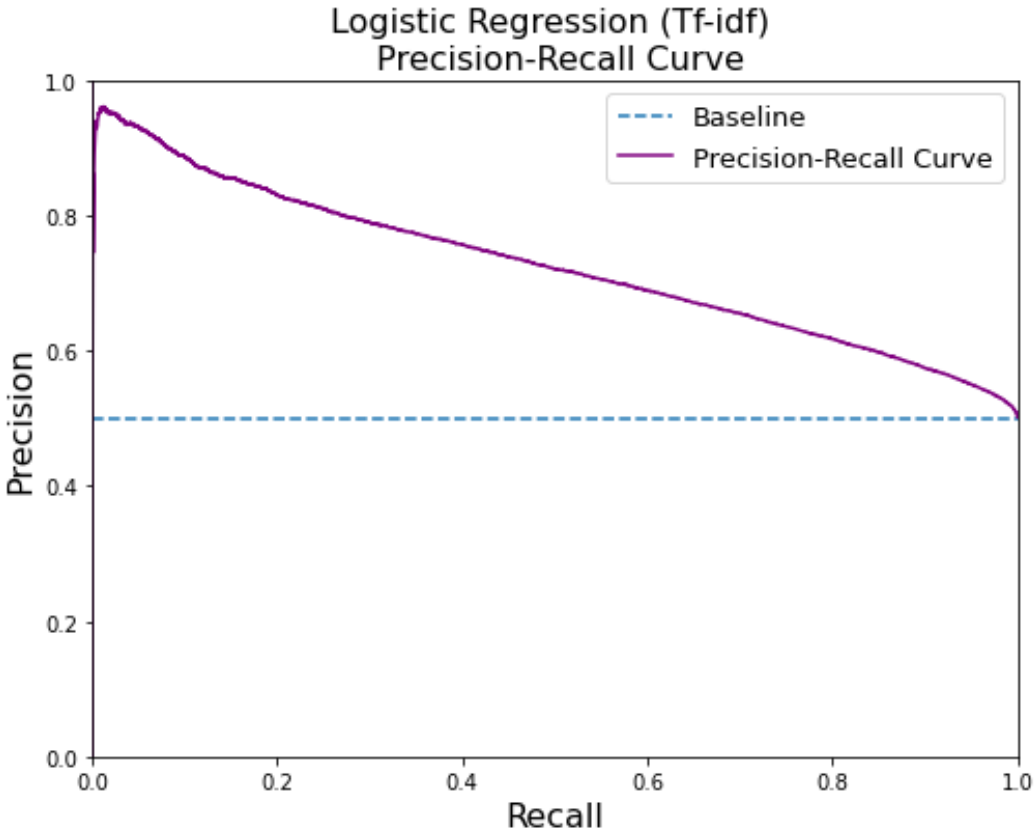
Accuracy and Precision Recall

One major advantage when measuring accuracy is that it is a straightforward method for comparing the performance between models; however, it has a major disadvantage in that it inherently applies equal weight to false positives and false negatives, and these are critical to recognize when dealing with data in the real world. As such, also evaluating a model based on precision and recall is an important practice to follow. Precision proves worthwhile as it helps to minimize false positives. This is achieved by calculating a ratio of true positives over all

positives. Recall is similar but its utility is in reducing the impact of false negatives. Recall calculates a ratio of the true positives over the sum of true positives and false negatives
As it relates to the specific data and context of this project, false negatives are far more important to seek out and avoid, as compared to false positives. This is because a false positive would signify that a realistically simple document would be mislabeled as 'difficult.' A nonnative speaker would not have trouble in this case. But with a false negative, a document that is supposedly 'simple,' but, is difficult to read, would be much more troubling to the individual, and is a headache we should try to avoid.

Sklearn offers useful plotting tools for understanding the precision-recall relationship, and we've plotted these charts below. Precision-recall curves and confusion matrices are useful metrics to check regarding evaluation of our models' performance. A confusion matrix provides great utility by summarizing a model's correct predictions along with the type 1 and type 2 errors made. The numbers which are output and displayed within a confusion matrix represent the numbers used during the calculation of precision and recall. Precision-recall curves are very helpful as they provide a visual representation of a model's trade-offs regarding precision compared to recall, each on different axes. With perfectly distributed, cross validated data (which likely never happens with real world data), the precision-recall curve would approach the upper right corner, meaning that both precision and recall have been maximized. As the precision-recall curve (charted below) generated from our model originates near the upper left limit and remains above baseline in the right quadrant, there is a strong to moderate degree of both precision and recall.


Confusion Matrix
Logistic Regression (Tf-idf)
28037
13427
14443
27241
0
1
True label
0
1
Predicted label
28000
26000
24000
22000
20000
18000
16000
14000


Logistic Regression (Tf-idf)
Precision-Recall Curve
Baseline
Precision-Recall Curve
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0

Failures

We tested several methods, parameters, and models. As stated earlier, we created over 200 models in the supervised learning section. The accuracy of each of the models tested on the validation set was over 65% except for the first Naïve Bayes model we ran without adjusting the default parameters. The highest accuracy for the validation data was slightly above 70% for the logistic regression model. We explored many options for exceeding the 70% threshold. We do believe it is possible considering the Decision Tree method was trained, albeit while overfitting, to nearly 98% accuracy. However, no method was capable. We do not feel the models were flawed but the data included too much variability. When we explored the data to examine the distribution, we looked at individual tokens to see how they were classified. On many occasions a single token would be classified as simple 50% of the time and 50% as not simple. When the data is 50/50 the outcomes can be no better. The perceived complexity didn't really follow any patterns. Eggs, for example, was classified as both 1 and 0 as was the word farm. So we couldn't use word length as a parameter. We also couldn't exclude non-alphanumeric characters. Those too had varying classifications.

As we discussed earlier, the Naïve Bayes method did present some problems when we tried the Gaussian classifier. The Gaussian NB Method required the dense matrix and was did not work with the sparse Tf-idf and Count Vectorized data arrays. The computational resources were not available to make the method work when we tried to convert the data to arrays as directed by the error instructions. The Gaussian Naïve Bayes method wasn't the only thing that failed us. We tried on several occasions to apply feature reduction techniques to the data so that we could examine the associated numerical values but were unsuccessful in implementing them. We were able to use the max_features function of the tf-idf and count vectorizers. However, the vectorizers don't provide a method for examining values such as principal components. We did attempt to apply the Principal Component Analysis to the data but, like the Naïve Bayes, the computer ran through the memory is a short amount of time. We liked the idea of PCA because we thought we might be able to set the number of features based on the component values. In the end, we had to settle on the percentage of features function we explained above.

One process that failed, much to our dismay, was our attempt to build our own tf-idf vectorizer. The tf-idf vectorizer performs a lot of data cleaning in the background to produce its feature names. It performs functions such as removing all the punctuation and converting the token to lowercase. However, our data had many instances of punctuation having an actual value the reader classified as not simple. That means punctuation matters in our model. We also assumed the more the vectorizer cleaned the more we limited its ability to classify. So, we tried to recreate the tf-idf vectorizer to keep the punctuation and just split the data based on a space delimiter. The mathematics of our function was not incorrect. We tested it against on a short corpus of data. However, when we tried to apply the full training set against our function, the computer never had enough memory to finish.

One thing we learned in the process is we have limited knowledge on how to reduce the complexity of a computation. We repeatedly ran into situations where there wasn't enough memory, or the computation took several hours to run. We realized we have limited knowledge on how to conduct distributed computing or how to run the computations on GPU versus CPU. We did try Dask as a method for addressing this issue, but Dask cannot run on the full set of scikit classifiers. Dask was a great method for distributed computing but just limited in its breath. We also had some success with scikit-learn-intelex to accelerate the applications and we did not deviate from its use once we discovered it. However, it is a behind the scenes applications that provides no insight into how it works and no options for setting different parameters.

Unsupervised Learning

We explored the underlying structure of the Wikipedia data (detailed above) using Latent Dirichlet Allocation (LDA). LDA is a generative topic model that assumes each topic generates words based on their probability distribution and that each document is produced as a mixture of some topics. In other words, a document is represented by a distinct topic distribution and a topic is represented by a distinct word distribution. Given a corpus of documents, LDA attempts to work backwards to determine which topics would create the documents.

To implement our LDA model, our team employed the following pipeline: loading an array of cleaned and preprocessed training data, vectorizing that data using a Count Vectorizer as is appropriate for probabilistic models, fitting those vectors to our LDA model, and evaluating the results. We were interested to see the documents grouped together visually by topic. Some questions we hoped to answer were: *Do certain topics / themes emerge from the given corpus? Can we use topics uncovered from our unsupervised learning to improve results in our supervised learning?*

Methods

We attempted to tune parameters using a grid search and altering the number of components and learning decay parameters. Our best model would be where log-likelihood was maximized, perplexity was minimized, and our topics were coherent and understandable. Luckily, sklearn provides a class model_selection.GridSearchCV that takes as input a parameter grid and results with the estimator that was chosen by the search, i.e. that estimator that gave the highest score.

Ideally, we would have been able to run the grid search on the full set of training data; however, due to computational constraints, we were forced to take a sampling of the training data. With the grid parameters below and a random sampling of 2.5% of the training data, our search returned an ideal number of components to be 5 and the optimal learning decay rate of .9.

```
param_grid={'learning_decay': [0.5, 0.7, 0.9],
            'n_components': [5, 10, 15, 20]})
```

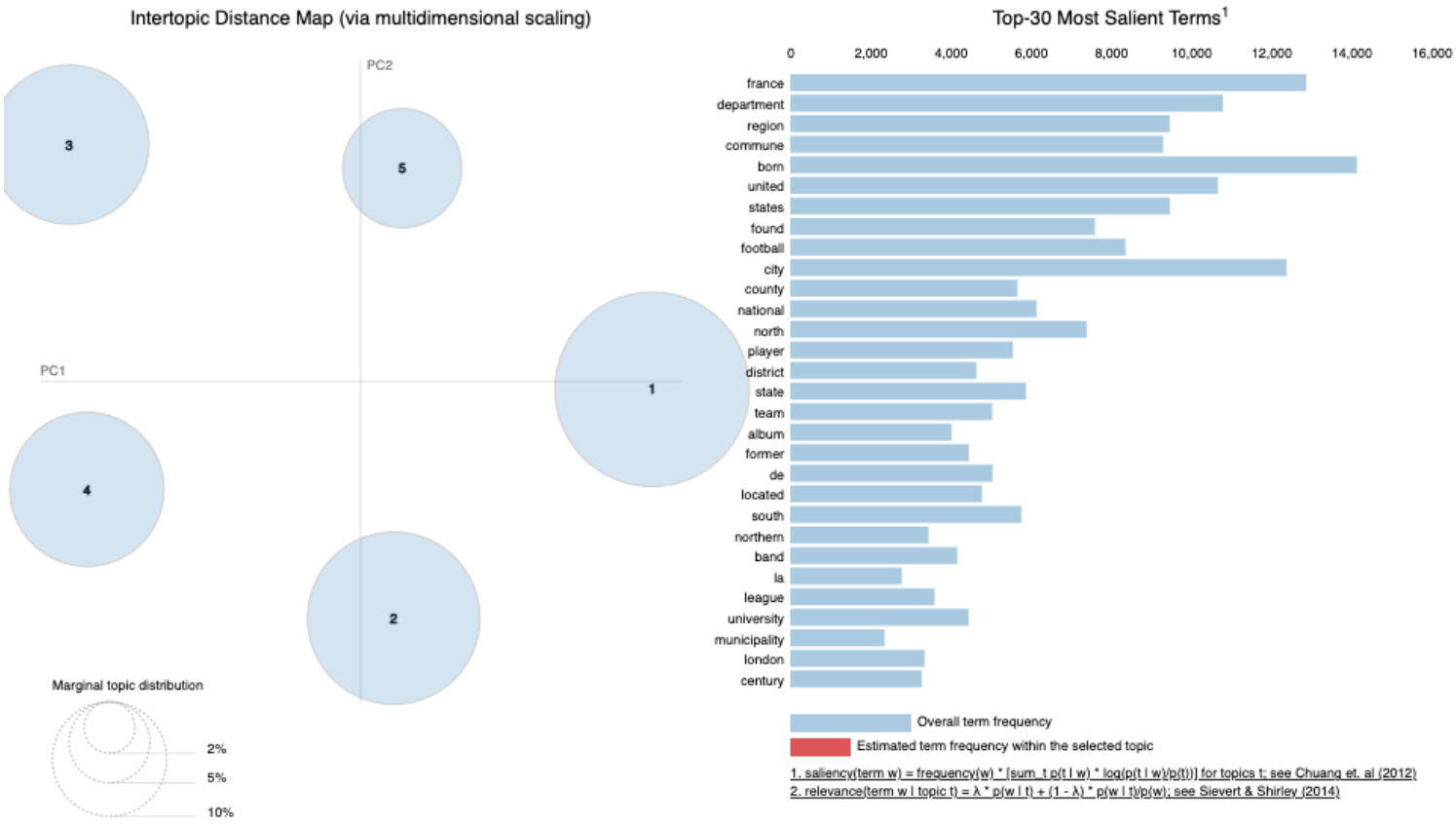
```
Best Model's Params:  {'learning_decay': 0.9, 'n_components': 5}
Best Log Likelihood Score:  -223871.07622527666
Model Perplexity:  12628.567756397826
```

With those tuned parameters, we reran our model with the full training set, n_components = 5, learning_decay = .9, and n_jobs = -1 to produce the below topic model with log likelihood of -32,687,799 and a perplexity score of 10,269. We decided to visualize our data using TSNE as this provided the greatest intertopic distance for our model.

```
Log Likelihood:  -32687798.603710953
Perplexity:  10269.149707010338
{'batch_size': 128,
 'doc_topic_prior': None,
 'evaluate_every': -1,
 'learning_decay': 0.9,
 'learning_method': 'batch',
 'learning_offset': 10.0,
 'max_doc_update_iter': 100,
 'max_iter': 10,
 'mean_change_tol': 0.001,
 'n_components': 5,
 'n_jobs': -1,
 'perp_tol': 0.1,
 'random_state': None,
 'topic_word_prior': None,
 'total_samples': 1000000.0,
 'verbose': 0}
```

We encountered the most challenges while attempting to tune our parameters because LDA modeling is such a computationally expensive task. Running a grid search is time and resource intensive and future LDA work with this data would benefit from the utilization of multiple cores at that stage. As stated, some of the ways we sought to remedy these challenges were using a smaller sampling of our data, setting n_jobs to -1 to make use of all processing power, and limiting the grid search size (including only four options for number of topics and three options for learning decay).

Unsupervised Evaluation



When broken out into five components, clusters maintain good inter-cluster distance. Despite this, topic coherence was still difficult to distill. Our attempt at generalizing and giving names to topics can be found below:

- Topic 1: Production (use, time, made, work, show)
- Topic 2: Football (football, player, club, team, national)
- Topic 3: History (century, first, city, known, history)
- Topic 4: Music (song, called, album, time, band)
- Topic 5: France (France, region, district, province, north)

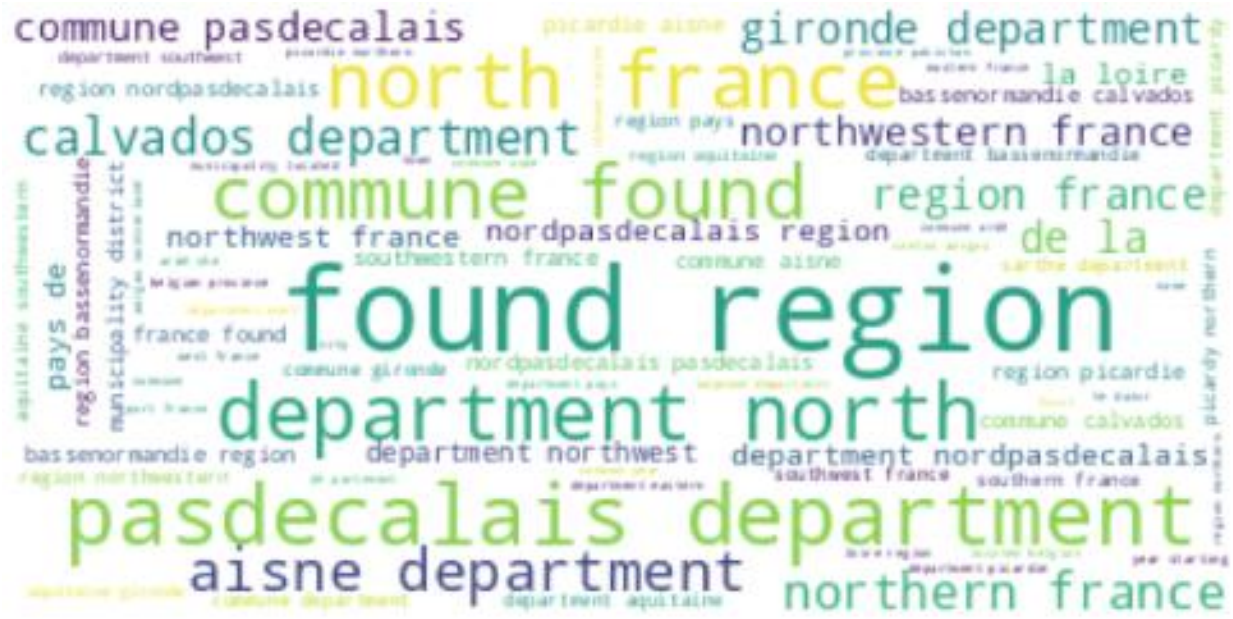
The resulting word clouds for each can be seen below.

Topic 1: Production

Topic 2: Football

Topic 3: History

Topic 4: Music



Topic 5: France

Future modeling attempts can improve on our parameter tuning by taking advantage of more time and computing power to evaluate more combinations of parameters (n_components = 5, 6, 7, 8, etc.) to better fine-tune parameters. Attempts to do so and attempts to tune parameters on the entire training data set, were unsuccessful. This was due to our team's lack of experience running jobs on multiple cores and topic modeling with data of this size and scope.

While we relied on some quantitative metrics to evaluate our models, we could have relied on additional quantitative metrics and human comprehension to further evaluate our models. In future modeling, research teams can add a human comprehension test at each step of parameter tuning to identify the ideal number of topics present in the documents. Additionally, a good model will generate topics with high topic coherence scores, which measures the similarity between high scoring words in each topic. Future research would also benefit from the measure of coherence scores.

Beyond topic modeling, future research teams can take advantage of other forms of unsupervised learning to uncover other interesting patterns in the text data such as k-means clustering in addition to other methods of topic modeling such as Latent Semantic Analysis or Non-negative Matrix Factorization.

Finally, our team faced challenges incorporating all aspects of the unsupervised learning methods (topic model parameter choice, wordcloud creation, LDA visualization output, etc.) directly into our Makefile on Github. With more time and knowledge of utilizing virtual environments we believe we could achieve this; however, due to time constraints we instead

decided to add interactive Python notebooks to our repository and focus on the content of our project.

Discussion

Part A

We highlighted many of the things we learned throughout this report. One of the important things we learned in Part A is our lack of understanding of "big data". We don't have a full grasp of how to perform distributed or parallel computing. We don't understand how to optimize our functions to reduce the burdens of computation. We know this takes exposure to learn and often takes a lot more time to learn and explore than a single month. It was sobering, though, to see how limited we were.

One of the surprising things is simple is sometimes best. One of the oldest and most tested methods, Logistic Regression, reigned supreme.

We wouldn't extend our supervised solutions from a modeling perspective. We would focus more on trying to address the randomness inherent in the data. We didn't have the insight to change values in the label field. We considered changing the values at the onset of the project but looked back on our own experiences learning foreign languages. For example, one of the project members studied Arabic as a minor in their undergraduate work. That member once took an hour to look up the roots of each of four consecutive words which took over an hour. It turns out the four consecutive words was someone's name. Words like Henry may seem simple to English speakers who understand it is a name, but to an English as a second language learner Henry is abstract. Who are we to say what is simple and what is not.

Part B

While working through Part B, our team learned a lot about topic modeling. First, we learned how time and resource intensive a proper unsupervised topic modeling evaluative process can be. Second, we were surprised at how few topics our evaluation determined was best given how difficult it was to summarize / identify topics given the list of most frequent terms. Finally, we were also constantly reminded of the importance, and iterative process, of data cleaning especially when working with text data. Text data is messy, especially when that text data is scraped from Wikipedia and the process of cleaning, tokenizing, and lemmatizing it was certainly not one-and-done. With more time and resources, the iterative process would continue; new inaccuracies in the cleaned data would be discovered and corrected.

As stated above, extensions of Part B of our work would include utilizing multiple cores to speed up unsupervised topic modeling, which would allow us to test even more hyperparameters; exploring other unsupervised methods including K-means clustering, Latent Semantic Analysis,

and Non-negative Matrix Factorization, among others; and include human comprehension at each step of the parameter tuning process to ensure that topics make sense to human audiences.

A more streamlined, concise, and clean codebase for Parts A & B could also be achieved with additional time and resources.

Ethical Considerations

In Part A, generalizing the perceived difficulty of text data may not actually be translatable to non-native speakers and may create frustration for some non-native speakers trying to practice or acquire English as a secondary language. I.e., our model could state that a text body should be 'easy' to digest, but some individuals may struggle with acquiring new language or just certain sentence structures, so this could be offensive and lead to frustration.

As for Part B, without the intervention of human actors, topic models can further perpetuate harmful language and amplify hurtful word associations, especially when using data from non-moderated sites. Models are only as unbiased as the data we use to train them. If the data contain harmful language that further reinforce stereotypes about gender, race, sexual orientation, age, ability, etc. and those biases are not identified prior to topic modeling, we risk further perpetuating and amplifying those stereotypes. Ensuring that these tropes do not exist in the underlying data, either through crowd-sourced moderation of the incoming data or human evaluation of the resulting topic terms themselves, must be viewed as essential work.

Statement of Work

Most work was collaborative and shared across team members. Group members were all present on twice-weekly calls to discuss progress and identify roadblocks. Members' areas of focus are detailed below.

- Tyler Allerton: Exploratory data analysis, data cleaning, feature extraction, supervised learning support, final write-up
- James Beck: Exploratory data analysis, supervised learning methods, evaluation, & visualizations
- Kevin Borah: Exploratory data analysis, unsupervised learning methods, evaluation, & visualizations
- Joint: Code optimization, code simplification, Git management

Bibliography

Guido, A. C. (2017). *Introduction to Machine Learning with Python.* Sebastopol: O'Reilly Media.

Scikit. (2022, October 1). *scikit-learn*. Retrieved from scikit-learn: https://scikit-learn.org/stable/supervised_learning.html#supervised-learning