Ian MacDonald

CIS 530

Homework 2 Writeup

All Complex Baseline

Model Performance (Training)

Precision: 0.43275

Recall: 1.0

F-Score: 0.604083057058105

Model Performance (Development)

Precision: 0.418

Recall: 1.0

F-Score: 0.5895627644569816

Word Length Baseline

Thresholds Tried: [3,4,5,6,7,8,9]

Threshold Chosen: 7

Model Performance (Threshold:7, Training)

Precision: 0.6007401315789473

Recall: 0.8440207972270364

F-Score: 0.7018976699495555

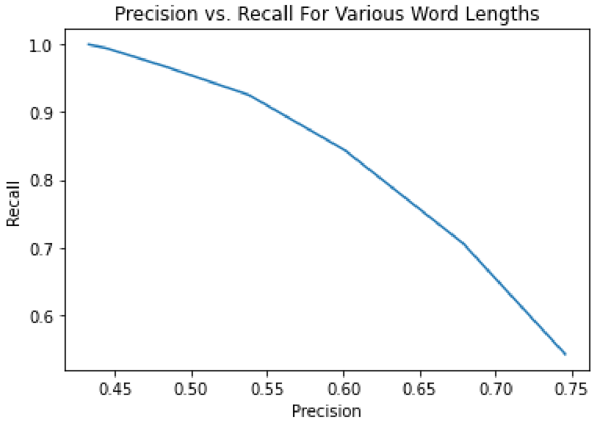
Model Performance (Threshold:7, Development)

Precision: 0.6053511705685619

Recall: 0.8660287081339713

F-Score: 0.7125984251968505

Precision-Recall Curve



Word Frequency Baseline

avg_freq = Average Word Frequency, std_freq = Standard Deviation of Word Frequencies

Thresholds Tried: 1000 evenly spaced increments between (avg_freq+(std_freq*.1)) and (avg_freq+(3*std_freq))

Threshold Chosen: 19902502.348709133

Model Performance (Training)

Precision: 0.5654785742891469

Recall: 0.8157134604274986

F-Score: 0.6679280983916746

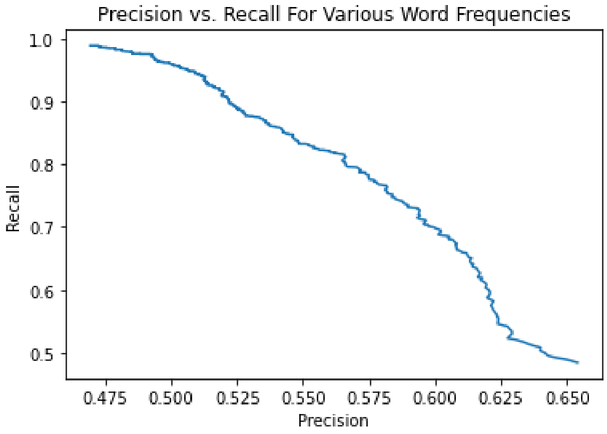
Model Performance (Development)

Precision: 0.556782334384858

Recall: 0.8444976076555024

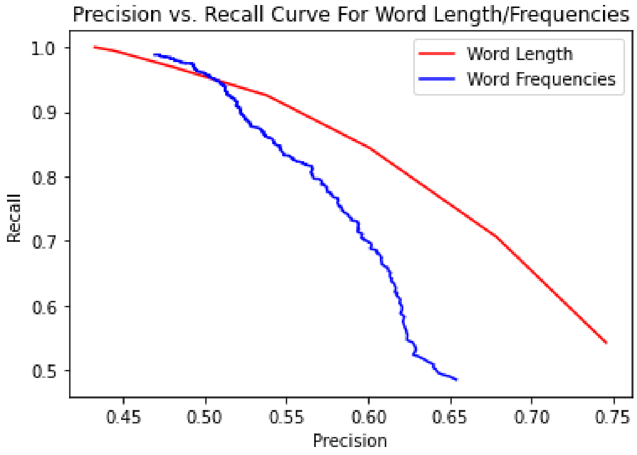
F-Score: 0.6711026615969581

Precision-Recall Curve



Word Length Baseline vs. Word Frequency Baseline Comparison

Precision-Recall Curve



Analysis

On average, the word length classifier seems to perform better than the word frequency classifier. As shown in the graph above, as precision increases recall decreases for both classifiers, but decreases less steeply for the word length classifier compared to the word frequency classifier.

Naïve Bayes Classifier

Model Performance (Training)

Precision: 0.4950379451255108

Recall: 0.9797804737146159

F-Score: 0.6577467519875897

Model Performance (Development)

Precision: 0.46929316338354576

Recall: 0.9688995215311005

F-Score: 0.6323185011709602

Logistic Regression Classifier

Model Performance (Training)

Precision: 0.7250159134309357

Recall: 0.6580011554015021

F-Score: 0.689884918231375

Model Performance (Development)

Precision: 0.7268170426065163

Recall: 0.69377990430622

F-Score: 0.7099143206854344

Naïve Bayes Classifier vs. Logistic Regression Classifier

Analysis

The logistic regression classifier performed better on this data set than the naïve bayes classifier, with and F-Score of .7099 vs .6323 on the development data set for each classifier, respectively. The naïve bayes model had good recall but poor precision, indicating that it was predicting words to be complex far too often, performing similarly to the all complex baseline. The logistic regression found a much better balance between precision and recall, which is why it's F-Score was higher. Naïve bayes makes certain assumptions, such as conditional independence of features, which may not have been met, resulting in poor performance compared to logistic regression.

Final Model

Features

- Word Length
- Word Frequency
- Number of Syllables
    - Calculated using the given syllables.py helper file.
- Word Probability
    - Word Probability calculated by taking each individual letter in the word, and multiplying together the probability of seeing each letter based on the relative frequency the letter appears in text (based on data from https://en.wikipedia.org/wiki/Letter_frequency) . Results are then scaled accordingly based on number of letters in each word, to account for longer words having lower probabilities on average due to more letters. Intuition behind this is that complex words may be more likely to be made up of uncommon letters.
- Part of Speech

- One hot encoded, calculated using wordnet. If word has multiple part of speech options, each possible option is marked as true.

Classifier

Chosen Classifier: Adaboost

Adaboost performed best on training data, as well as on development data, and thus was chosen as the classifier used in the final model.

Model Performance (Training)

Precision: 0.7495601173020527

Recall: 0.7383015597920277

F-Score: 0.7438882421420255

Model Performance (Development)

Precision: 0.7341176470588235

Recall: 0.7464114832535885

F-Score: 0.7402135231316725

Model Analysis

The final model performed well on the development set, with an F-Score of .7402, beating the F-Scores of all other classifiers tried as well as all the baselines. However, the F-Score was still nowhere near perfect, meaning there is room for improvement, possibly in tuning the model parameters, extracting better features, or both. Below are some words the model performed well on, some it performed poorly on, and possible improvements that can be made.

Performed Well – Short, non-complex words

Examples:

Word: part, Prediction: 0, Actual: 0

Word: bear, Prediction: 0, Actual: 0

Word: snacks, Prediction: 0, Actual: 0

Word: loss, Prediction: 0, Actual: 0

The model performed very well on short words. However, a vast majority of words that were small (3,4,5 letters) were not complex, which meant the model would have pretty good success predicting not complex for most of the words, which it did. However, the model did not just solely predict not complex for these words, as it had some hits, such as:

Word: adorns, Prediction: 1, Actual: 1

For the most part though, it was tripped up a lot by short, complex words, which leads into the next section -

Performed Poorly – Short, Complex Words

As stated above, the model leaned heavily towards predicting not complex for short words, which meant that it had problems with short, complex words. Some examples:

Word: seize, Prediction: 0, Actual: 1

Word: salvo, Prediction: 0, Actual: 1

Word: opted, Prediction: 0, Actual: 1

Word: scam, Prediction: 0, Actual: 1

It was difficult to pinpoint what exactly made these words complex, and why the model got them wrong. They were common enough that the word frequency feature didn't cause the model to predict complex, and the letters for the most part are pretty common. Even in the word seize, which contains the least common letter z, it still contains the most common letter e, and twice at that. Intuitively, I would still categorize these words as complex, but it was hard to define why, and thus hard to add/adjust features accordingly.

Performed Poorly – Complex Words with Common Letters

Similar to the reason the model missed on the word seize, even though it contains an uncommon letter, the model missed on some complex words that contained a lot of common letters, such as:

Word: tenants, Prediction: 0, Actual: 1

Word: cottage, Prediction: 0, Actual: 1

Word: relics, Prediction: 0, Actual: 1

Word: gathered, Prediction: 0, Actual: 1

These words contain lots of vowels, which are the most common letters, and some of the most common consonants (r,s,t,n,h, etc.). The word probability feature could have thrown the model off on these words (even though I would argue a word like gathered should be considered non-complex), as it may be understating how often complex words are made up of common letters.

Possible Improvements

It seems that the word probability feature may be too heavily influenced by certain letters, i.e. having two or more high probability letters (e, a, etc.) may be leading too often to false negatives, and vice versa for false positives. An improvement may be to adjust this feature to some sort of Boolean feature indicating just the presence or not of certain letters. Another improvement could be to perform parameter tuning on the AdaBoost model. Currently, it's using the default parameters provided by scikit-learn, but tuning these parameters would likely lead to a better model.

Other Models Tried

Classifiers and Performances (Tried, but ultimately not chosen as final model)

Model Performance (Random Forest, Training)

Precision: 0.7238758708043065

Recall: 0.6603119584055459

F-Score: 0.6906344410876133

Model Performance (Random Forest, Development)

Precision: 0.7270408163265306

Recall: 0.6818181818181818

F-Score: 0.7037037037037036

Model Performance (SVM, Training)

Precision: 0.7192982456140351

Recall: 0.7105719237435009

F-Score: 0.7149084568439408

Model Performance (SVM, Development)

Precision: 0.7294685990338164

Recall: 0.722488038277512

F-Score: 0.7259615384615384