[redacted]

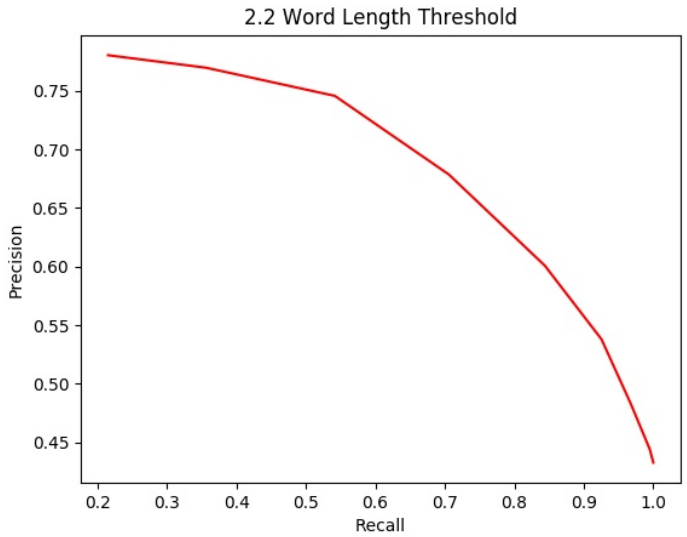
Report for Homework 2

I. Basics

Our results for the basic models are as follows:

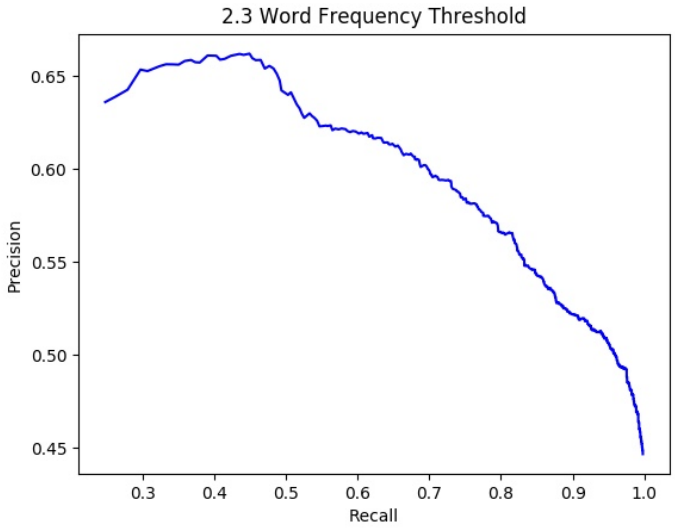
Classifier	Train			Dev		
	Precision	Recall	F-score	Precision	Recall	F-score
All-complex	0.433	1.000	0.604	0.418	1.000	0.590
Word-length Baseline	0.601	0.844	0.671	0.605	0.866	0.713
Word-freq Baseline	0.566	0.816	0.668	0.557	0.844	0.671
Naive Bayes	0.495	0.980	0.658	0.469	0.969	0.632
Logistic Regression	0.725	0.658	0.690	0.727	0.694	0.710

1. For the word-length baseline model, we have tried every threshold value between [1,12]. We find that P-R values are approximately inversely related, and that the f-scores lie in a concave curve with the best threshold 7. See the following figure:

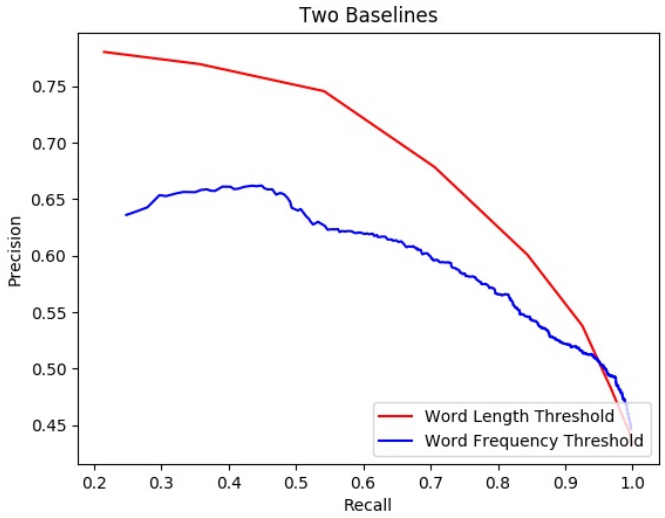


2. For the word-frequency baseline model, we have tried threshold values in the range [1000000, 100000000] with a granularity of 100000. Likewise, in approximation, P and R are negatively related, with the highest f-score achieved when the threshold is within

the range from 19,984,000 to 19,900,000. The plot is as follows:



3. In comparison to Word-freq thresholding, it seems that the Word-length thresholding model works better on average:



It is visible that for every R value, the Word-length model almost always has a higher corresponding P than the Word-freq model, except for a few extreme values on the tail.

II. Our model

1. Classifiers tried:

Apart from the Logistic Regression Classifier and the Naive Bayes Classifier, we have

tried Support Vector Machines with both Linear and Gaussian Kernels, Random Forest and AdaBoost. AdaBoost turns out to have performances consistently better than the other classifiers using the baseline features:

Classifier	F-score on Train	F-score on Dev
Logistic Regression	0.690	0.710
Naive Bayes	0.658	0.632
AdaBoost	**0.741**	**0.747**
Random Forest	0.945	0.656
SVM with Linear Kernel	0.696	0.709
SVM with RGB Kernel	0.691	0.712

2. Features tried:

Our best classifier includes the following features:

1. word length
2. word frequency in Google Ngram
3. number of syllables
4. ratio of vowel
   (following the intuition that a word like 'Copacabana' might sound easier to parse than 'kryptonite'. For the purposes of the function, hyphens are treated as vowels, since they don't constitute a symbol which is difficult to parse.)
5. word frequency in SubtlexUS corpus (Brysbaert, M. & New, B., 2009)

For each one in the above list, we have also tried to include its correspondent feature based on the lemmatized word. We use the lemmatizer from NLTK, adding an additional parameter of POS in order to improve its precision. After multiple attempts, we find that the incorporation of three 'lemmatized' features (1, 4, and 5) along with the previous five 'bare' ones gives us the best performance:

Best Classifier	Train			Dev			**Test**
	P	R	F	P	R	F	F
AdaBoost	0.784	0.782	0.783	0.752	0.792	0.772	**0.786**

Such a fact is somewhat counterintuitive, because some seemingly important features (e.g. the lemmatized version of 2) actually result in a lower f-score. Anyway, for optimal performance on the current task, we decide to eliminate them from our model.

There are also several features we have tried additionally, but have failed to see improvements:

1. average word length in the sentence
2. average word frequency in the sentence
3. length of the sentence
4. log frequency instead of raw frequency
5. Number of Wordnet synsets

3. Tweaks to the program:

We have noticed that the original load_ngram_count() function provided to us is simplified (probably for the sake of efficiency), because it only takes in tokens with an initial letter in lowercase. But that underestimates certain counts, since there are false positive examples like "university" (whose capitalized counterpart "University" has a significantly higher frequency). Therefore, we modify the load_ngram_count() function by first transforming the token into lowercase, and then adding its frequency to the corresponding entry in the dictionary.

4. Error Analysis:

Here are some examples that our best model is good/not good at:

	Examples
True Complex	outstripped, psychological, indict, curriculums, doctorate, eons, enthralled, commissioners, infestation, eye-blink, engineering, eerie, hinge
True Simple	two-stage, tapping, businesses, movement, roughly, neighbors, cheating, accepted, marking, younger, 28-nation, attorney, thumbs-up, earliest
False Complex	threatened, attached, rulings, unexpected, environmental, tropics, granite, yawning, revealing, father-daughter, five-time, no-food-or-drink
False Simple	spreading, downed, whistling, essence, seize, improve, secure, regularly, helicopters, recommended, embrace, reduce, twists, boomed, oath, scam

The categories of words our model behaves poorly on include:

1. Words with inflections. There are still a number of inflected words which our model incorrectly recognizes as complex, for instance, "threatened", "attached", "yawning", "revealing". However, the problem turns out not to be on the lemmatizer (since it's actually doing well in lemmatizing these examples) but somewhere else, which is quite curious.

2. Compound words. Conspicuously, a considerable proportion of compound words are erroneously labelled as complex words, such as “father-daughter”, “five-time”, “no-food-or-drink” and so on. Our initial interpretation is that these compounds may have very low frequency in the counts file, so that in order to get rid of this bias, we tried to introduce a word splitter in our classifier. Surprisingly though, it turns out to lower our f-score, for which we have not been able to find a reasonable interpretation.