**Assignment 1: Text Classification (due 15 February 2019)**

The goal of this assignment is to build a classifier to classify simple vs. complex words. Simple words are words such as ‘heard’, ‘sat’ while complex words are ‘abdicate’, ‘liaison’, etc. This classification is part of a larger task in NLP called text simplification which replaces complex words with simpler synonyms to make the text more easily understood by younger readers, English as second language speakers, etc.

The goals of the assignment are:

1. To implement NLP evaluation methods (precision, recall, F1) by yourself in order to understand them better
2. To learn to do experiments in NLP that involves implementing simple baselines to understand how easy/difficult the task is, experimenting with a range of features and models using **sklearn**

The deliverables are:

1. Your code implementing functions in the skeleton file that is provided
2. The README file on how to run your code to produce answers to Q2, Q3, and Q4 below. **The grader must be able to run your code for your submission to be graded**.
3. Your model’s output for the test set using only the provided training and development data. You will be given the test set that does not have labels. You will build a classifier to predict the labels on the test set. Submit the labeled test set. We will score your predictions on the test set and create a leaderboard showing whose classifier produces the best performance.

The materials provided in this zip file are:

1. The skeleton code that contains the functions you should implement
2. Data sets that contains the training/development/test sets
3. Unigram counts from Google N-gram corpus
4. syllables.py, a script for counting the number of syllables in a word

This dataset was collected by crowdsourcing human judgements. Nine human annotators were asked to identify at least 10 complex words in each text. From here, words that were identified as complex by at least 3 annotators were labeled as complex. In addition, words that were identified as complex by zero annotators were labeled as simple. One thing to note is that they kept only nouns, verbs, adjectives, and adverbs, and removed stopwords and proper nouns. For this assignment, the words are split up into 3,000 words for training, 1,000 words for development, and 1000 words for testing.

Shown below is an example of the training data. The test data does not include the label column:

| **WORD** | **LABEL** | **ANNOTATORS** | **SENTENCE** | **SENTENCE\_INDEX** |
| --- | --- | --- | --- | --- |
| paths | 0 | 0 | The Cannery project will feature drought-tolerant landscaping along its bike paths , and most of the front yards will be landscaped with low-water plants in place of grass . | 10 |
| banks | 0 | 0 | Extending their protests into the workweek , Hong Kong democracy activists continued occupying major thoroughfares Monday , forcing the closure of some schools , banks and other businesses in the semi-autonomous Chinese territory . | 24 |
| fair-weather | 1 | 5 | Months ago , many warned him not to invest in a place where fair-weather tourists flee in the fall and the big lake ‘s waters turn cold and storm-tossed , forcing the 100 or so hardy full-time residents of Cornucopia to hibernate for the winter . | 13 |
| krill | 1 | 7 | But unlike the other whales , the 25-foot-long young adult stuck near the surface – and it did n’t dive down to feast on the blooms of krill that attract humpbacks to the bay . | 27 |

Here is what the different fields in the file mean:

* WORD: The word to be classified
* LABEL: 0 for simple words, 1 for complex words
* ANNOTATORS: The number of annotators who labeled the word as complex
* SENTENCE: The sentence that was shown to annotators when they labeled the word as simple or complex
* SENTENCE\_INDEX: The index of the word in the sentence (0 indexed, space delimited).

The function load\_file(data\_file), which takes in the file name (data\_file) of one of the datasets, and reads in the words and labels from these files has been implemented in the skeleton code.

**Q1. Implement Evaluation Functions (6 points)**

Consider complex words as positive examples and simple words as negative examples. Implement these functions in the skeleton code:

* get\_precision(y\_pred, y\_true)
* get\_recall(y\_pred, y\_true)
* get\_fscore(y\_pred, y\_true)

where y\_pred is the list of predicted labels from a classifier and y\_true is the list of gold labels. You **must** write your own code to calculate these, do not use sklearn or other package’s built-in functions for this. You will use these functions to evaluate your classifier in this assignment.

**Q2. Implement Baselines (15 points)**

1. Majority class baseline.   
   Implement a majority class baseline which always predicts the majority class and is one of the simplest classifier. Complete the function all\_complex(data\_file), which takes in the file name of one of the datasets, labels each word in the dataset as complex, and returns out the precision, recall, and f-score.

Call the function and print out the precision, recall, and f-score on the training data and the development data individually.

1. Word length baseline.   
   Use a slightly more complex baseline, the length of each word, to predict its complexity. Try various thresholds for word length to classify them as simple or otherwise. For example, you might set a threshold of 7, meaning that any words with less than 7 characters will be labeled simple, and any words with 7 characters or more will be labeled complex. Once you find a threshold value that performs the best on the training data (based on F1 score that tries to balance between good precision and good recall), use that same threshold value on the development data.   
     
   Complete the function word\_length\_threshold(training\_file, development\_file). Call this function, taking in both the training and development data files, and print out the precision, recall, and f-score for your best threshold’s performance on the training and development data individually.
2. Word frequency baseline.   
   Use another baseline, the frequency of each word, to predict its complexity. This classifier is similar to the word length classifier but thresholds on word frequency instead of length. The frequencies are provided from unigram counts in Google N-Gram corpus. The function load\_ngram\_counts(ngram\_counts\_file) to load these counts into a dictionary in Python has been provided.   
     
   Complete the function word\_frequency\_threshold(training\_file, development\_file, counts), where counts is the dictionary of word frequencies. This function again returns the precision, recall, and f-score for your best threshold’s performance on both the training and development data.

Call this function, taking in both the training and development data files and counts, and print out the precision, recall, and f-score for your best frequency threshold’s performance on the training and development data individually.

**Q3. Implement Classifiers (15 points)**

1. Use the built-in **Naïve Bayes** classifier from sklearn (<https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html>) to build your simple vs. complex word classifier. Sklearn classifiers take in numpy arrays instead of regular lists. For example, to create a numpy list of length 5, you can use the following commands:

>>> import numpy as np

>>> X = np.array([1,2,3,4,5])

To train a classifier, you need two numpy arrays: **X\_train** (an **m** by **n** array where **m** is the number of words in the dataset and **n** is the number of features for each word); and **Y** an array of length **m** for the labels of each of the words. Given these two arrays, we can fit a Naïve Bayes classifier with the following commands:

>>> from sklearn.naive\_bayes import GaussianNB

>>> clf = GaussianNB()

>>> clf.fit(X\_train, Y)

to use the classifier to predict labels for a set of words, you need another numpy array: **X\_test** (an **m’** by **n** array where **m’** is the number of words in the test dataset and **n** is the number of features for each word, same as the number of features in **X\_train**). The use the following to predict labels:

>>> Y\_pred = clf.predict(X\_test)

Complete the function naive\_bayes(training\_file, development\_file, counts) that will train a Naive Bayes classifier on the *training* data using word length and word frequency as features, and returns your model’s precision, recall, and f-score on the training data and the development data individually.

Call this function and print out the precision, recall, and f-score on the training data and the development data individually.

NOTE: Before training and testing a classifier, it is generally important to normalize your features. This means that you need to find the mean and standard deviation (sd) of a feature. Then, for each row, perform the following transformation:

X\_scaled = (X\_original - mean)/sd

Be sure to always use the means and standard deviations from the training data.

1. Implement the same classifier as the above, but using **Logistic Regression** from sklearn (<https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html>).

Complete the function logistic\_regression(training\_file, development\_file, counts) that will train a Logistic Regression classifier on the *training* data, and returns your model’s precision, recall, and f-score on the training data and the development data individually.

Call this function and print out the precision, recall, and f-score on the training data and the development data individually.

**Q4. Implement Your Own Model (25 points)**

You will build your own classifier for the complex word identification task, and compare your results to that of your classmates.

You can choose any other types of classifier, and any additional features you can think of! For classifiers, beyond Naive Bayes and Logistic Regression, you might consider trying SVM, Random Forests, among others. Additional word features that you might consider include number of syllables, number of WordNet synonyms, and number of WordNet senses. For counting the number of syllables, a python script syllables.py contains the function count\_syllables(word), which you may use. To use WordNet in Python, refer to this [documentation](http://www.nltk.org/howto/wordnet.html). You could also include sentence-based complexity features, such as length of the sentence, average word length, and average word frequency.

When trying different classifiers, we recommend that you train on training data, and test on the development data, like the previous sections.

To receive full credit, you **must** try at least 1 type of classifier (not including Naive Bayes and Logistic Regression), and at least two features (not including length and frequency).

Train your ***best*** model on both the training and development data. Call this classifier to predict and print out labels for the test data. Submit these labels in a text file named test\_labels.txt (with one label per line); be sure NOT to shuffle the order of the test examples.

**Q5. Bonus points (max 5 points)**

+3 Top 10 on the leaderboard/+5 Top 3 on leaderboard. The performances of the baselines will be included on the leaderboard. In order to receive full bonus credit, your model must be able to outperform all of the baselines.

**Recommended Readings**

|  |
| --- |
| [Naive Bayes Classification and Sentiment.](https://web.stanford.edu/~jurafsky/slp3/6.pdf) Dan Jurafsky and James H. Martin. Speech and Language Processing (3rd edition draft) . |
| [Logistic Regression.](https://web.stanford.edu/~jurafsky/slp3/7.pdf) Dan Jurafsky and James H. Martin. Speech and Language Processing (3rd edition draft) . |
| [Problems in Current Text Simplification Research: New Data Can Help.](http://www.cis.upenn.edu/~ccb/publications/new-data-for-text-simplification.pdf) Wei Xu, Chris Callison-Burch, and Courtney Napoles. TACL 2015. |
| [Comparison of Techniques to Automatically Identify Complex Words.](http://aclweb.org/anthology/P/P13/P13-3015.pdf) Matthew Shardlow. ACL 2013. |
| [SemEval 2016 Task 11: Complex Word Identification.](https://www.researchgate.net/profile/Gustavo_Paetzold/publication/305334627_SemEval_2016_Task_11_Complex_Word_Identification/links/57bab70a08ae14f440bd9722/SemEval-2016-Task-11-Complex-Word-Identification.pdf) Gustavo Paetzold and Lucia Specia. ACL 2016. |