This assignment is due on Friday, September 24, 2021 before 11:59PM. This assignment may be done with a partner.

Text Classification : Assignment 2

For this assignment, we'll be building a text classifier. The goal of our text classifer will be to distinguish between words that are simple and words that are complex. Example simple words are *heard, sat, feet, shops,* and *town*, and example complex words are *abdicate, detained, liaison,* and *vintners*. Distinguishing between simple and complex words is the first step in a larger NLP task called text simplification, which aims to replace complex words with simpler synonyms. Text simplification is potentially useful for re-writing texts so that they can be more easily understood by younger readers, people learning English as a second language, or people with learning disabilities.

The learning goals of this assignment are:

- Understand an important class of NLP evaluation methods (precision, recall and F1), and implement them yourself.
- Employ common experimental design practices in NLP. Split the annotated data into training/development/test sets, implement simple baselines to determine how difficult the task is, and experiment with a range of features and models.
- Get an introduction to sklearn, an excellent machine learning Python package.

We will provide you with training and development data that has been manually labeled. We will also give you a test set without labels. You will build a classifier to predict the labels on our test set. You can upload your classifier's predictions to Gradescope. We will score its predictions and maintain a leaderboard showing whose classifier has the best performance.

Here are the materials that you should download for this assignment:

- Skeleton code (hw2_skeleton.py) - this provides some of the functions that you should implement.
- Data sets (data.tar.gz) - this is a tarball with the training/dev/test sets.
- Unigram counts (http://www.cis.upenn.edu/~cis530/18sp/data/ngram_counts.txt.gz) from the Google N-gram corpus (https://research.googleblog.com/2006/08/all-our-n-gram-are-belong-to-you.html).

This assignment has several deliverables:

- Your implementations for the functions in the skeleton code.
- Your model's output for the test set (your model will be ranked on a leaderboard against the other students' outputs)
- Your writeup about the homework. Here is an example writeup (example_writeup.pdf) that illustrates what we are looking for.

Identifying Complex Words

Automated text simplification is an NLP task, where the goal is to take as input a complex text, and return a text that is easier to understand. One of the most logical first steps in text simplification, and example of text classification, is identifying which words in a text are hard to understand, and which words are easy to understand.

We have prepared a labeled training set for this assignment. We provide a dataset of words and their corresponding sentences that has been split into training, development, and test sets. The training set is disjoint, so if a word appears in the training set, it will not also appear in the test set or the development set.

This dataset was collected by taking the first 200 tokens in 200 complex texts, and crowdsourcing human judgements. We asked nine human annotators 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 we kept only nouns, verbs, adjectives, and adverbs, and removed stopwords (i.e. common words like `the` or `and`) and proper nouns. After this filtering, we were left with 5,922 unique words. For this homework, we split these words up into 4,000 words for training, 1,000 words for development, and the remaining 922 words are reserved for testing.

Shown below is an example of the training data. Note that the training data and development data files have the same formatting, and the test data does not include the label column:

WORD	LABEL	ANNOTATORS	SENTENCE	SENTENCE_INDEX
jumping	0	0	`` Coleman with his jumping frog – bet stranger $ 50 – stranger had no frog & C got him one – in the meantime stranger filled C 's frog full of shot & he could n't jump – the stranger 's frog won . ''	4
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
affirmed	1	8	CHARLESTON , S.C. – A grand jury affirmed the state of South Carolina 's murder charge on Monday against a white former North Charleston police officer who fatally shot an unarmed black man trying to run from a traffic stop .	7

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).

We have provided 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. CLARIFICATION: You should make sure your load_file() function makes every word lowercase. We have edited the skeleton to do so as of 1/18.

Note: While the context from which each word was found is provided, you do not need it for the majority of the assignment. The only time you may need this is if you choose to implement any context-based features in your own classifier in Section 4.

1. Implement the Evaluation Metrics

Before we start with this text classification task, we need to first determine how we will evaluate our results. The most common metrics for evaluating binary classification (especially in cases of class imbalance) are precision, recall, and f-score. For this assignment, complex words are considered positive examples, and simple words are considered negative examples.

For this problem, you will fill in the following functions:

- `get_precision(y_pred, y_true)`
- `get_recall(y_pred, y_true)`
- `get_fscore(y_pred, y_true)`

Here, `y_pred` is list of predicted labels from a classifier, and `y_true` is a list of the true labels.

You may **not** use sklearn's built-in functions for this, you must instead write your own code to calculate these metrics. You will be using these functions to evaluate your classifiers later on in this assignment.

We recommend that you also write a function `test_predictions(y_pred, y_true)`, which prints out the precision, recall, and f-score. This function will be helpful later on!

2. Baselines

Implement a majority class baseline

You should start by implementing simple baselines as classifiers. Your first baseline is a majority class baseline which is one of the most simple classifier. You should 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.

Please report the precision, recall, and f-score on both the training data and the development data individually to be graded.

Word length baseline

For our next baseline, we will use a slightly complex baseline, the length of each word to predict its complexity.

For the word length baseline, you should try setting various thresholds for word length to classify them as simple or otherwise. For example, you might set a threshold of 9, meaning that any words with less than 9 characters will be labeled simple, and any words with 9 characters or more will be labeled complex. Once you find the best threshold using the training data, use this same threshold for the development data as well.

You will be filling in the function `word_length_threshold(training_file, development_file)`. This function takes in both the training and development data files, and returns out the precision, recall, and f-score for your best threshold's performance on both the training and development data.

Usually, Precision and Recall are inversely related and while building binary-classification systems we try to find a good balance between them (by maximizing f-score, for example). It is often useful to plot the Precision-Recall curve for various settings of the classifier to gauge its performance and compare it to other classifiers. For example, for this baseline, a Precision-Recall curve can be plotted by plotting the Precision (on the y-axis) and Recall (on the X-axis) for different values of word-length threshold.

In your write-up, please report the precision, recall, and f-score for the training and development data individually, along with the range of thresholds you tried. Also plot the Precision-Recall curve for the various thresholds you tried. For plotting, matplotlib (https://matplotlib.org/) is a useful python library.

Word frequency baseline

Our final baseline is a classifier similar to the last one, but thresholds on word frequency instead of length. We have provided Google NGram frequencies in the text file `ngram_counts.txt`, along with the helper function `load_ngram_counts(ngram_counts_file)` to load them into Python as a dictionary.

You will be filling in 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 fscore for your best threshold's performance on both the training and development data.

Please again report the precision, recall, and f-score on the training and development data individually, along with the range of thresholds you tried, and the best threshold to be graded. Similar to the previous baseline, plot the Precision-Recall curve for range of thresholds you tried. Also, make a third plot that contains the P-R curve for both the baseline classifier. Which classifier looks better *on average*?

Note: Due to its size, loading the ngram counts into Python takes around 20 seconds, and finding the correct threshold may take a few minutes to run.

3. Classifiers

Naive Bayes classification

Now, let's move on to actual machine learning classifiers! For our first classifier, you will use the built-in Naive Bayes model from sklearn, to train a classifier. You should refer to the online sklearn documentation (http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html) when you are building your classifier.

The first thing to note is that sklearn classifiers take in `numpy` arrays, rather than regular lists. You may use the online `numpy` documentation. To create a `numpy` list of length 5, you can use the following Python 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. Once we have these two arrays, we can fit a Naive Bayes classifier using the following commands:

```
>>> from sklearn.naive_bayes import GaussianNB
>>> clf = GaussianNB()
>>> clf.fit(X_train, Y)
```

Finally, to use your model to predict the labels for a set of words, you only need one `numpy` array: `X_test`, an `m'` by `n` array, where `m'` is the number of words in the test set, and `n` is the number of features for each word. Note that the `n` used here is the same as the `n` in `X_train`. Then, we can use our classifier to predict labels using the following command:

```
>>> Y_pred = clf.predict(X_test)
```

You should fill in the function `naive_bayes(training_file, development_file, counts)`. This function 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.

In your write-up, please report the precision, recall, and f-score on the training and development data for your Naive Bayes classifier that uses word length and word frequency.

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`.

Logistic Regression

Next, you will use sklearn's built-in Logistic Regression classifier. Again, we will use word length and word frequency as your two features. You should refer to the online sklearn documentation (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html) when you are building your classifier. To import and use this model, use the following command:

```
>>> from sklearn.linear_model import LogisticRegression
>>> clf = LogisticRegression()
```

For this problem, you will be filling in the function `logistic_regression(training_file, development_file, counts)`. This function 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.

Again, please report the precision, recall, and f-score on the training and development data.

Comparing Naive Bayes and Logistic Regression

After implementing Naive Bayes and Logistic Regression classifiers, you will notice that their performance is not identical, even though they are given the same data. Add a paragraph to your write up that discusses which model performed better on this task.

4. Build your own model

Finally, the fun part! In this section, you will build your own classifier for the complex word identification task, and compare your results to that of your classmates. You will also perform an error analysis for your best performing model.

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`, `Decision Trees`, and `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, we have provided a python script syllables.py (syllables.py) that 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.

In your writeup, please include a description of all of the models and features that you tried. 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).

Note: You can also tune the parameters of your model, e.g. what type of kernel to use. This is NOT required, as some of you may not be that familiar with this.

Analyze your model

An important part of text classification tasks is to determine what your model is getting correct, and what your model is getting wrong. For this problem, you must train your best model on the training data, and report the precision, recall, and f-score on the development data.

In addition, need to perform a detailed error analysis of your models. Give several examples of words on which your best model performs well. Also give examples of words which your best model performs poorly on, and identify at least TWO categories of words on which your model is making errors.

Leaderboard

Finally, train your best model on both the training and development data. You will use this classifier to predict labels for the test data, and will submit these labels in a text file named `test_labels.txt` (with one label per line) to the leaderboard; be sure NOT to shuffle the order of the test examples. Instructions for how to post to the leaderboard will be posted on Piazza soon.

The performances of the baselines will be included on the leaderboard. In order to receive full credit, your model must be able to outperform all of the baselines. In addition, the top 3 teams will receive 5 bonus points!

(Optional) Leaderboard using outside data

While the training data we have provided is sufficient for completing this assignment, it is not the only data for the task of identifying complex words. As an optional addition to this homework, you may look for and use any additional training data, and submit your predicted labels in a text file named `test_labels.txt` to a separate leaderboard.

As a start, we recommend looking at the SemEval 2016 dataset (http://alt.qcri.org/semeval2016/task11/), a dataset that was used in a complex words identification competition. In addition, you can try to use data from Newsela (https://newsela.com). Newsela's editors re-write newspaper articles to be appropriate for students at different grade levels. The company has generously shared a dataset with us. The Newsela data **may not** be re-distributed outside of Penn. You can find the data on eniac at `/home1/c/ccb/data/newsela/newsela_article_corpus_with_scripts_2016-01-29.1.zip` .

Good luck, and have fun!

5. Deliverables

Here are the deliverables that you will need to submit:

- Your code. This should implement the skeleton files that we provide. It should be written in Python 3.
- Your model's output for the test set using only the provided training and development data.
- (Optional) your model's output for the test set, using any data that you want.
- Your writeup in the form of a PDF.

6. Recommended readings

Naive Bayes Classification and Sentiment. (https://web.stanford.edu/~jurafsky/slp3/4.pdf) Dan Jurafsky and James H. Martin. Speech and Language Processing (3rd edition dra
Logistic Regression. (https://web.stanford.edu/~jurafsky/slp3/5.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. Abstract BibTex
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/Sem 2016-Task-11-Complex-Word-Identification.pdf) Gustavo Paetzold and Lucia Specia. ACL 2016.

Grading Rubric

This assignment was worth 60 points total. The rubic used for grading this homework is below.

Implement the Evaluation Metrics (5 points total)

1.1 (5) Functions `get_precision` , `get_recall` , and `get_fscore` correct.

Baselines (15 points total)

2.1 (5) Function `all_complex` correct

- -5 No results reported in writeup

2.2 (5) Word length threshold correct

- -1 Wrong threshold identified
- -1 Reported performance much higher than expected
- -1 No mention of threshold chosen in writeup

2.3 (5) Word Frequency threshold correct

- -2 Did not try a large enough range of thresholds
- -1 Large enough range tried, but best threshold reported is off by a lot
- -1 No mention of threshold chosen in writeup

Classifiers (15 points total)

3.1 (5) Naive Bayes classifier correct

- -2 Very low performance on development data
- -1 Did not report performance on training data

3.2 (5) Logistic Regression classifier correct

- -1 Did not report performance on training data

3.3 (5) Logistic Regression vs. Naive Bayes analysis correct

- -5 Missing
- -3 No analysis of why performances differ (Very common mistake!)

Build your own model (25 points total)

4.1 (10) Beat baselines (Everyone got this!)

4.2 (10) Complete model analysis

- -5 No description at all about features/model
- -2 No description of why features/model were chosen
- -2 Overfit model to training data

- -1 Stated that overfitting was ok in this task

4.3 (5) Error analysis correct

- -5 Missing
- -2 Examples given, but no categories of mistakes identified

Extra Credit (5 points max)

- +3 Top 10 on the mandatory leaderboard
- +5 Top 3 on either leaderboard

Reno Kriz and Chris Callison-Burch developed this homework assignment for UPenn's CIS 530 class in Fall 2018.