Assignment 1
CSE 143: Intro to Natural Language Processing

University of California Santa Cruz

This assignment is to be done in Python 3 without use of any NLP libraries (such as NLTK).

Your final writeup for this assignment, including the problem and the report of the experimental findings of the programming part, should be no more than four pages long. Most of your grade will depend on the quality of the report. You should submit it as a PDF. We *strongly* recommend typesetting your scientific writing using LaTeX. Some free tools that might help: Overleaf (online), TexLive (cross-platform), MacTex (Mac), and TexStudio (Windows).

1 Problem: Naive Bayes (33%)

In this first problem, you will implement a Naive Bayes classifier by hand (without programming) for sentiment analysis.

A collection of movie reviews (data D) contains keywords and binary labels for whether each review was positive ($+$) or negative ($-$). The data is shown below: for example, the cell at the intersection of "Review 1" and "epic" indicates that the text of Review 1 contains 2 tokens of the word "epic".

Answer the following questions, reporting all scores as **log-probabilities**, to three significant figures.

Review	great	amazing	epic	boring	terrible	disappointing	Y
1	2	2	2	1	1	0	+
2	1	4	0	0	0	1	+
3	3	2	3	1	0	0	+
4	0	1	1	2	1	2	-
5	1	0	2	1	2	2	-
6	1	0	1	2	1	2	-

(a) Assume that you have trained a Naive Bayes model on data D to detect positive vs. negative movie reviews. Compute the model's predicted scores for both positive and negative classes for the following sentence S (i.e. $P(+|S)$ and $P(-|S)$), and determine which label the model will apply to S. Use only the keywords listed in the table above.

S:

```
   The film was great, the plot was simply amazing! Makes other
superhero movies look terrible, this was not disappointing.
```

(b) The counts in the original data are sparse and may lead to overfitting, e.g. a strong prior on assigning the negative label to reviews that contain "terrible". What would happen if you applied *smoothing* (discussed in section and in the readings)? Apply *add-1 smoothing* and recompute the Naive Bayes model's predicted scores for S. Did the predicted label change?

(c) What is an additional feature that you could extract from text to improve the classification of sentences like S (not necessarily in Naive Bayes), and how would it help improve the classification?

2 Programming: Movie Review Sentiment Classification

Preliminaries

In this problem, you will do binary text classification of movie reviews. You need to both answer the questions and submit your code.
We provide a dataset of polar 'positive' and 'negative' movie reviews extracted from the Large Movie Review Dataset: https://ai.stanford.edu/ amaas/data/sentiment/ [1]
This problem will require programming in Python 3. The goal is to build a Naive Bayes model and a logistic regression model that you learned from the class on a real-world classification dataset for movie reviews. Finally, you will explore how to design better features and improve the accuracy of your models for this task.

The dataset you will be using is collected from IDMB movie reviews online. Each example is labeled as 1 (a positive review) or 0 (a negative review). To get started, you should first download the data and starter code from Canvas.

Try to run:

```
python main.py --model AlwaysPredictZero
```

This will load the data and run a default classifier `AlwaysPredictZero` which always predicts label 0 (a negative review). You should be able to see the reported train accuracy = 0.4993 (699/1400). That says, always predicting a review is negative isn't that good. Let's try to build better classifiers!

Note that for *your* solutions, you need to implement models **without use of any machine learning packages such as `sklearn`**. Your solutions should use simple Python lists or `numpy` arrays (`numpy` arrays are recommended since they are much faster than Python lists.)

However, to have a quick check with your implementations, you can compare the accuracy between the models you have implemented and related models in `sklearn` packages on the train, dev, and test sets. If you want, you can try `sklearn` in non-submitted code, to get an idea of what the expected outputs should be and how much training time your models should require.

2.1 Naive Bayes (33%)

In this part, implement a Naive Bayes model with *add-1 smoothing*, as discussed in the section and in the readings. You are required to implement the `NaiveBayesClassifier` class in `classifiers.py`.

You will probably want to take a look at the `UnigramFeature` class in `utils.py` that we have implemented for you already.

After you finish coding, run `python main.py --model NaiveBayes` to check the performance.

Deliverables In the write-up, be sure to fully describe your models and experimental procedure. Provide graphs, tables, charts or other summary evidence to support any claims you make.

1. Report the accuracy on the training, dev, and test sets.
2. Look at the output of your model on some randomly selected examples in the dev set. What do you observe? Why do you think the model has predicted the way it has for those specific examples?
3. List the 10 words that, under your model, have the highest ratio of $P(w|1)/P(w|0)$ (the most distinctly positive words) and list the 10 words with the lowest ratio. What trends do you see?

2.2 Logistic Regression (33%)

In this part, you will implement a Logistic Regression model. You are required to implement the `LogisticRegressionClassifier class` in `classifiers.py`. First, implement a logistic regression model without regularization and run `python main.py --model LogisticRegression`, compare the performance with your Naive Bayes approach. Next, we would like to experiment with L2 regularization. Add L2 regularization with different weights, such as $\lambda = \{0.0001, 0.001, 0.01, 0.1, 1, 10\}$.

Deliverables In the write-up, be sure to fully describe your models and experimental procedure. Provide graphs, tables, charts or other summary evidence to support any claims you make.

1. Report the accuracy without L2 regularization on the train, dev, and test sets. How does it compare to Naive Bayes?
2. Add L2 regularization with different weights, such as $\lambda = \{0.0001, 0.001, 0.01, 0.1, 1, 10\}$.
   In your writeup, describe what you observed.

2.3 Bonus (10%)

In the last part, you'll explore and implement a more sophisicated set of features. You need to implement the class `BigramFeature` or modify the class `CustomFeature` in `utils.py`. Here are some common strategies (you are welcome to implement one or more of them or come up with others!):

1. Remove stopwords (e.g. a, the, in),
2. Use a mixture of unigrams, bigrams or trigrams, (It may take exponentially more time to run bigram features than unigram features. For example, you can choose to implement bigrams based on Naive Bayes model, since it takes much less time than Logistic Regression.)
3. Use TF-IDF (refer to http://www.tfidf.com/) features.

Use your creativity for this problem and try to obtain an accuracy as high as possible on your dev set, and evaluate how it performs on the test set! We will call the BonusClassifier, which should simply specify which classifier to use, and the CustomFeature class which should contain any custom features you'd like to be used for this part.

Deliverables In the write-up, be sure to fully describe your models and experimental procedure. Provide graphs, tables, charts or other summary evidence to support any claims you make.

1. Describe the features you implemented, the model you decided to use for this part and your reasoning for this. Report the accuracy on the training, dev, and test sets.

In your write-up, describe the features you implemented, the model you decided to use for this part and your reasoning for this.

Submission Instructions

Submit a zip file (`A1.zip`) on Canvas, containing the following:

- **Code**: You code should be implemented in Python 3, and needs to be runnable. Submit your code together with a neatly written README file to instruct how to run your code with different settings. We assume that you always follow good practice of coding (commenting, structuring), and these factors are not central to your grade. However, please provide well commented code if you want partial credit. If you have multiple files, provide a short description in the preamble of each file.
- **Report**: As noted above, your writeup should be four pages long or less, in PDF (one-inch margins, reasonable font sizes). Include the names of the members in your group at the top of the report. Part of the training we aim to give you in this class includes practice with technical writing. Organize your report as neatly as possible, and articulate your thoughts as clearly as possible. We prefer quality over quantity. Do not flood the report with tangential information such as low-level documentation of your code that belongs in code comments or the README. Similarly, when discussing the experimental results, do not copy and paste the entire system output directly to the report. Instead, create tables and figures to organize the experimental results.

References

[1] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P11-1015.