Kevin Lin

Problem Set 1: Sentiment classification with Naive Bayes and other techniques

In this problem set, you will be experimenting with different classifiers to distinguish positive movie reviews from negative movie reviews. The classifiers are:

* A simple keyword classifier. (Remember: do not use rule-based keyword approaches like this in your project!)
* An unsmoothed Naive Bayes classifier.
* A smoothed Naive Bayes classifier.
* A pre-trained classifier from the `textblob` library if you are running the code on your own computer.

This is due Friday, February 21, at 11:59pm EST.

YOU WILL FAIL THIS PROBLEM SET IF YOU DO NOT READ THIS README.

Getting started:

Option 1
Clone your repo down to your own machine. You will find a python program, `nb.py`, which you will be modifying and running on your own computer.

Option 2
Follow [this link to a Colab version of the `nb.py` program](https://colab.research.google.com/drive/1_gpP0qj0G130_dypwDzfA9oGzzNovGOi?usp=sharing). This is a good option if you aren't used to administering Python and managing libraries on your own computer, but it will involve a lot of "restart and run all". The notebook explains how to get the data into Colab.

Regardless of which option you choose you are required to submit a PDF with answers to all the questions in this README!

There are three directories of files in this repo (and in the repo you will clone on Colab, if you chose that option):

* `pos`: 900 positive movie reviews, one review per file
* `neg`: 900 negative movie reviews, one review per file
* `test`: 100 negative movie reviews and 100 positive movie reviews, one review per file

Feel free to look at the files in `pos` and `neg`, but *do not look at the files in* `test`.

Q1: Run the program `nb.py` (or do `Run all` in Colab), and report the accuracy metrics it prints out. No method has actually been implemented, so everything is just a majority class baseline where everything is marked as negative. What is the majority class baseline accuracy of this system? (1 point)

The majority class baseline accuracy of the system is 0.5 or 50%

Part A: User-provided keyword classifier (6 points in total)
Visually inspect the files in the `pos` and `neg` directories (but do not look at `test`!). Identify 10 words that seem strongly associated with positive reviews and 10 words that seem strongly associated with negative reviews. In `nb.py` (or in your Colab notebook), go to where it says `YOUR PART A CODE STARTS HERE`, and add your keywords to the two lists, `negative_keywords[]` and `positive_keywords[]`.

Q2: Run the program `nb.py` again, and adjust your keywords until you get classification accuracy over 0.5. Report the accuracy of this classifier and report how many times you had to adjust the keyword lists to improve upon the random baseline.

I only had to adjust the keyword lists once and was able to get a keyword baseline that was 0.66 for user keyword accuracy.

Part B: Naïve Bayes classifier (12 points in total)
In class, we discussed using a naïve Bayes classifier for spam filtering and for word sense disambiguation. Here, you'll be building your own implementation of a naïve Bayes classifier for a real dataset of movie reviews.

Recall that in naïve Bayes, you need to consider two probabilities: (1) the probability of a word, given a positive or negative review, and (2) the prior probability of each class. The prior probability of each class in our data is the same (0.5) because the two classes have the same number of examples (900) so you can ignore the prior for this problem set.

In `nb.py` (and the Colab notebook), I've provided a lot of code. For Part B, you just need to write the function that calculates the probability of a word in each of the two classes (positive and negative). Find the place in `nb.py` (or the Colab notebook) where it says `YOUR PART B CODE STARTS HERE`, and follow the instructions.

When you have implemented your code, run the program `nb.py` (or in Colab, `Run all`) again. The Naive Bayes classifier should return an accuracy above 0.7. If it doesn't, you didn't do it correctly, so try again.

Q3: Which has higher accuracy: your keyword classifier or the naïve Bayes classifier? Why do you think one is better than the other?

The naive bayes classifier has a better accuracy of 0.725 versus the 0.66 from my classification. I think this is because I was only able to quantify and count the frequency of the words that I visually saw the most and considered "positive". On the other hand, the naive bayes classifier considers all words and computes their relative probability for that class and is much more encompassing in terms of the words being considered.

Q4: Write some temporary code in nb.py (or your Colab notebook) so that you can find three test reviews that were misclassified by either of the two classifiers, and report those reviews here.

This was inserted into the calculate_accuracy() function

```
if filepolarity == naive_bayes(reviewwords):
        nbcorrect += 1
    else:
        print(f"File is classified as: {filepolarity}")
        print(f"File is classified by NB as: {naive_bayes(reviewwords)}")
        print(reviewwords)
```

If the formal classification doesn't match the NB classification, it prints out the actual classification and the class produced by the naive_bayes() function.

The NB classifier sometimes misclassifies reviews because it looks at words individually without considering their context. In these examples, it mistakenly labeled negative reviews as positive when they contained a few positive words and vice versa. This happens because the model doesn’t understand sarcasm, negation, or the overall tone—it just counts word frequencies. As a result, words like “good” or “best” can lead to incorrect positive classifications, while words like “disastrous” or “missing” can cause false negatives, even if the overall review sentiment is different.

Example #1
File is classified as: neg
File is classified by NB as: pos
['know', 'special', 'action-packed', 'takes', 'pointless', 'story', 'proof', 'boring', 'left', 'key', 'man', 'century', 'galaxy', 'confusing', 'rip-off', 'makes', 'effects', 'legendary', 'tell', 'saving', 'details', 'come', 'survivors', 'fine', 'kale', 'animation', 'japan', 'named', 'get', 'race', 'spaceship', 'movies', 'star', 'much', 'young', 'aliens', 'one', 'animated', '31st', 'despite', 'e', 're-generating', 'human', 'blaster', 'payoff', 'double-crosses', 'basic', 'good', 'total', 'unknowingly', 'clue', 'process', 'fights', 'take', 'apparent', 'universe', 'unmotivated', 'banter', 'main', 'wars', 'movie', 'early', 'supposed',.....

Example #2
File is classified as: pos
File is classified by NB as: neg

['mostly', 'sympathy', 'oh', 'missing', 'missteps', 'tells', 'men', 'in-laws', 'nurse', 'name', 'smoke', 'playing', 'fool', 'father', 'solving', 'lucky', 'groom-wannabe', 'beautiful', 'simple', 'overwhelmed', 'meeting', 'smoothly', 'guy', 'tug-of-war', 'water', 'get', 'fianc', 'immediately', 'also', 'asks', 'four', 'profession', 'disastrous', 'funny', 'good', 'humiliate', ...

Example #3
File is classified as: neg
File is classified by NB as: pos
['horror', 'pay', 'john', 'hysterical', 'importantly', 'parts', 'us', 'limited', 'ecstatic', 'name', 'playing', 'hand', 'chalk', 'adult', 'best', 'production', 'fortunately', 'simple', 'ever', 'raja', 'classroom', 'vartan', 'specifically', 'need', 'new', 'led', 'enrolling', 'presence', 'levels', 'full', 'get', 'class', 'occurring', 'teacher', 'also', 'written', 'business', 'sexual', 'otherwise', 'good', 'total', 'unlimited', 'found', 'sun', 'love', 'reilly', 'appreciate',

Q5: Why do you think those examples were not correctly classified? What might you do to improve this?

These examples were correctly classified by the Naive Bayes classifier since the NB classifier doesn't consider the context in which each individual token is being used under. For instance I saw that a fair amount of the incorrectly classified words contains the word "better", that was used in a negative context, but since the word "better" was used really frequently in positive reviews, it automatically classified the review as positive despite the word being used in a negative context. We could improve this by considering the context and the previous words by creating bigram tokens that consider the words around each token and whether the meaning was "negated" in a sense and the sentiment being reversed.

Part C: Smoothed Naive Bayes Classifier (18 points)
Find the place in `nb.py` (or your Colab notebook) where it says `YOUR PART C CODE STARTS HERE`, and follow the instructions to implement +1 smoothing. Your accuracy should continue to improve. If it doesn't, you did something wrong, so go back and try again.

Q6: Write some temporary code in nb.py (or your Colab notebook) so that you can find three reviews that were misclassified by unsmoothed Naive Bayes but correctly classified by smoothed Naive Bayes, and report those reviews here.

Using the temporary code:

```
if (filepolarity != naive_bayes(reviewwords)) and (filepolarity == smooth_naive_bayes(reviewwords)):
        print(f"File's correct classification is {filepolarity}")
        print(f"NB classifies it as {naive_bayes(reviewwords)}")
        print(f"Smooth NB classifies it as {smooth_naive_bayes(reviewwords)}")
        print(reviewwords)
```

We get the following outputs:

Example #1
File's correct classification is neg
NB classifies it as pos
Smooth NB classifies it as neg
['fistfights', 'human', 'get', 'creators', 'possesses', 'hollywood', 'story', 'e', 'details', 'makes', 'spaceship', 'unmotivated', 'fights', 'animation', 'named', 'one', 'early', 'key', 'basic', 'proof', 'dredge', 'come', 'kale', 'search', 'race', 'interesting', 'universe', 'banter', 'animated', 'good', 'premise', 'special', 'century', 'plain', 'wars', 'movies', 'take', ...

Example #2
File's correct classification is neg
NB classifies it as pos
Smooth NB classifies it as neg
['turns', 'intent', 'conference', 'machines', 'ouch', 'talking', 'amazing', 'comes', 'get', 'point', 'drift', 'generation', 'within', 'inspired', 'another', 'fifteen', 'courtesy', 'eating', 'blind', 'already', 'perfection', 'breathing', 'glitch', 'earful', 'name', 'find', 'computers', 'aunt', 'scores', 'one', 'involved', 'donated', 'minute-long', 'moments', 'quirky', 'let', 'satire', 'sperm', 'bank', 'compelling', "inkpot's", 'insecurities'....

Example #3
File's correct classification is neg
NB classifies it as pos
Smooth NB classifies it as neg
['graphics', 'complete', 'budget', 'q', 'criminal', 'different', 'laurene', 'comes', 'nowhere', 'whim', 'kinky', 'interest', 'impression', 'mystery', 'chases', 'shapely', 'misogynistic', 'splits', 'adheres', 'charles', 'made-for-tv', 'probably', 'differs', 'fall', 'three', 'rocky', 'alternately', 'whose', 'norma', 'resemblance', 'replaced', 'hammers', 'take', 'mafia', 'almost', 'laid', 'page-turner', 'improbable', 'appeared', 'conduct', 'alan', 'authority', 'yanked', 'mid-eighties', 'chambers', 'assured'...

Q7: Why do you think smoothing was able to improve classification performance?

I think smoothing was able to improve classification performance because it helps avoid giving too much weight to rare or unseen words in the training data. Without smoothing, the model might heavily penalize any words that don't appear frequently, treating them as highly unreliable or irrelevant. This can lead to overfitting, especially when uncommon words are only present in certain categories and might unfairly influence the classification. Smoothing ensures that even rare words are given a small, non-zero probability, preventing them from being entirely disregarded. This allows the model to generalize better by balancing the influence of common and uncommon words, ultimately improving its ability to classify new, unseen data accurately.

****Q8: Find three reviews that were still classified incorrectly, even with smoothing. Write them out here, and then comment on why you think they might have been challenging for the classifiers.****

Using the following temporary code:

```
if (filepolarity != smooth_naive_bayes(reviewwords)):
        print(f"File's correct classification is {filepolarity}")
        print(f"Smooth NB classifies it as {smooth_naive_bayes(reviewwords)}")
        print(reviewwords)
```

Example #1
File's correct classification is pos
Smooth NB classifies it as neg
['tommy', 'decides', 'even', 'one', 'mr', '17', 'alot', 'around', 'old', 'career', 'pick', 'directoral', 'soup', 'pestering', 'capable', 'sevigny', 'buscemi', 'starts', 'lounge', 'chloe', 'year', 'brother', 'debut', 'take', 'flick', 'movie', 'loser', 'directing', 'put', 'finding', 'michael', 'trees', 'tries', 'gave', 'dogs', 'cream', 'writing', 'local', 'reach', 'reservoir', 'think', 'acting', 'know', 'uncle', 'hanging', 'slow', 'expectation', 'well', 'girl', 'favorite', 'love', 'says', 'actors', 'ice', 'better', 'drive', 'seen', 'memorable', 'director', 'truck', 'obvious', 'awfully', 'steve', 'excellent', 'performence', 'liked', 'job', 'fargo', 'hand', 'good', 'bar', 'story', 'interest', 'performences', 'thinking', 'named', 'dies', 'debbie']

Example #2
File's correct classification is neg
Smooth NB classifies it as pos
['decides', 'crown', 'reasons', 'change', 'starting', 'result', 'matter', 'unique', 'long', 'asimov', 'since', 'crew', 'take', 'audience', 'awards', 'wrote', 'technology', 'academy', 'always', 'film', 'unlikely', '"bicentennial"', 'dared', 'gave', 'beginning', '50', 'gem', 'pursuit', 'leagues', 'potential', 'neill', 'ever', 'mrs', 'talented', 'perfect', 'destiny', 'circuits', 'sea', 'next', 'mushy', 'science', 'incredibly', 'sort', 'century', 'mom', 'dishes', 'require', 'today', 'failure', 'accepts', 'sentimentality', 'shows', 'minimum', 'children', 'emotional', 'entertainment', 'freedom', 'filmmakers', 'speaking', 'doubtfire', 'racial', '17th', 'williams', 'compare', 'make', 'culture', '20', 'story', 'big', 'difficult',.....

Example #3
File's correct classification is pos
Smooth NB classifies it as neg
['throughout', 'say', 'close', 'ding-dong', 'portrait', 'even', 'prominent', 'besides', 'one', 'comes', 'see', 'without', 'clear', 'whole', 'slows', 'kind', 'unfolds', 'critique', 'raping', 'us', 'proves', 'enough', 'shots', "florida's", 'managed', 'kissing', 'contract', 'really', 'specified', 'crawl', 'couple', 'continuing', 'many', "hollywood's", 'anything', 'added', 'facts', 'guidance', 'down-trodden', 'charges', 'long', 'chick', 'troopers', 'showing', 'ruining', 'pinup', 'russell', 'tad', 'scene', 'allowed', "o'", 'brother', 'florida', 'slick', ...

I think that the above reviews were a bit challenging for the classifiers because the lack of positive/negative words in the proper context. Specifically it seems like these examples had words that were neither positive nor negative, and the few instances where they were used were primarily used in a different context/meaning. Additionally some of these examples were just full of unique words, or words with typos causing the classifiers to consider them very unlikely for both cases and instead consistently fall back to the default probabilities for both cases.

Part D: Using the `textblob` library (6 points)
Find the place in `nb.py` (or your Colab notebook) where it says `YOUR PART D CODE STARTS HERE`, and follow the instructions for using the `textblob` library to determine the sentiment of a review. Once you have implemented this function, the results printed out for `textblob` accuracy will increase.

Here is a link to the `textblob` sentiment classification documentation:
http://textblob.readthedocs.io/en/dev/quickstart.html#sentiment-analysis

Please note that textblob is *not* a good sentiment classifier. You can use it as a baseline in your projects if you like, but it's not something you should rely on.

Q9: The instructions in the code tell you to use a threshold of 0. Experiment with changing that threshold to other values between -1 and 1. Which values improved accuracy? Which values decreased accuracy? How could you use the training data to select a threshold? (Note that changing the threshold like this is "cheating" since you are tuning to the test set.)

When I experimented with different threshold values between -1 and 1, I found that setting the threshold a little higher, around 0.10, gave me the best accuracy at 0.745. This was a pretty big improvement over the default threshold of 0.00, which had an accuracy of 0.63. Since the threshold decides whether a review is labeled “pos” (if the polarity is greater than the threshold) or “neg” (if it’s less), having a slightly higher threshold helped avoid classifying weakly positive reviews as positive when they might not be strongly positive. That small tweak seemed to make a noticeable difference.

But when I moved the threshold too much in either direction, accuracy dropped. Lowering it to -0.09 made accuracy go down to 0.505, probably because it started marking too many neutral or slightly negative reviews as positive. On the other hand, increasing it to 0.25 lowered accuracy to 0.52, which makes sense because it was being too strict and marking a lot of mildly positive reviews as negative. At the extreme ends: -1.00 and 1.00, accuracy was 0.50, which is basically random guessing.

To actually pick the best threshold properly, you’d want to use the training data and try different thresholds on a validation set instead of the test set. This way, you could find the best threshold before testing on completely unseen data. Adjusting the threshold

based on the test set is cheating, since you're tuning the model to work well on data it's technically not supposed to have seen yet. A better approach would be using cross validation to figure out the best threshold before final testing.

Part E: Reporting your results

Q10: Create a nicely formatted table of the accuracy of all of the classification options you explored in this problem: (1) random baseline, (2) user keywords, (3) naive Bayes, (4) smoothed naive Bayes, (5) `textblob` with 0 threshold, (6) `textblob` with improved threshold. (5 points)

Accuracy By Classification	
Classification Method	Accuracy
Random Baseline	0.50
User Keywords	0.66
Naive Bayes	0.725
Smoothed Naive Bayes	0.84
Textblob (0.00 threshold)	0.63
Textblob (0.10 threshold)	0.745

Add, commit, and push (1) your version of `nb.py` (or your Colab notebook) with all of the code you implemented for parts A, B, C, and D; and (2) a PDF containing the answers to all questions beginning with Q. If you used Colab, download and push the notebook AND share with me and the TAs. **This is due Friday, February 21 at 11:59pm EST.**