Tristan Basil

CS460G-001 Machine Learning – Dr. Brent Harrison

4/8/2018

Problem Set 5 - Report

This assignment implements naive Bayes to classify and generate text. There are two included programs: Program5.py, and Program5Bigram.py They are both implemented in Python 2. This report will provide a brief overview of each implementation and their results.

**Part 1: Classification using Naive Bayes and Unigram Tokens**

Program5.py implements a Bayesian network with unigrams and naïve Bayes to classify which Shakespearean character (Hamlet, Juliet, Macbeth, or Romeo) said which line. It is implemented in Python 2 and uses the following packages:

1. **sys** to get command line arguments (the filename),
2. **copy** to create deep copies of lists and vectors easily, and
3. **math** to calculate logarithms to prevent underflow.

For this part, Program4.py is run with the following command:

$ python Program5.py [hamletTrainFilename] [julietTrainFilename] [macbethTrainFilename] [romeoTrainFilename] [hamletTestFilename] [julietTestFilename] [macbethTestFilename] [romeoTestFilename]

Where each [\*trainFilename] should be replaced with the respective training file name and each [\*testFilename] should be replaced with the respective test filename. Note that each parameter must be passed exactly in this order to function properly. As an example, here is the command I used to run the program:

$ python Program5.py hamlet\_train.txt juliet\_train.txt macbeth\_train.txt romeo\_train.txt hamlet\_test.txt juliet\_test.txt macbeth\_test.txt romeo\_test.txt

This will initialize the Bayesian network with unigrams, then calculate the accuracy for each character, and the overall accuracy across every character. There were a few implementation decisions worth mentioning:

1. This implementation uses logarithms of probabilities to prevent underflow. Base 2 was chosen arbitrarily.
2. This implementation uses pseudocounts in the event that a given word is not found in that character’s train vocabulary but is found in another character’s.
3. In the event that a given word is not found in any character’s vocabulary, its probability is ignored, which helps prevent data from being skewed.

Here are the final accuracies yielded for each character with Program5.py:

* **Macbeth Accuracy**: 36.3636363636%
* **Hamlet Accuracy**: 51.3422818792%
* **Romeo Accuracy**: 40.7692307692%
* **Juliet Accuracy**: 51.5151515152%
* **Total Accuracy**: 46.1194029851%

**Part 2: Classification using Naïve Bayes and Bigram Tokens**

Program5Bigrams.py implements a Bayesian network with bigrams and naïve Bayes to classify which Shakespearean character (Hamlet, Juliet, Macbeth, or Romeo) said which line. It is an extension of Program5.py, is also implemented in Python 2 and uses the same packages as those listed for Part 1. It can be run with the following command:

$ python Program5Bigram.py [hamletTrainFilename] [julietTrainFilename] [macbethTrainFilename] [romeoTrainFilename] [hamletTestFilename] [julietTestFilename] [macbethTestFilename] [romeoTestFilename]

Where each [\*trainFilename] should be replaced with the respective training file name and each [\*testFilename] should be replaced with the respective test filename. Note that each parameter must be passed exactly in this order to function properly. As an example, here is the command I used to run the program:

$ python Program5Bigram.py hamlet\_train.txt juliet\_train.txt macbeth\_train.txt romeo\_train.txt hamlet\_test.txt juliet\_test.txt macbeth\_test.txt romeo\_test.txt

This will initialize the Bayesian network with bigrams, then calculate the accuracy for each character, and the overall accuracy across every character. Afterward, it will execute the extra credit portion of the assignment, which will be detailed in the next section of this report. As this is an extension of Program5.py, all implementation decisions detailed in Part 1 also apply to Program5Bigram.py. There were no additional implementation decisions worth mentioning for this part.

Here are the final accuracies yielded for each character with Program5Bigram.py:

* **Macbeth Accuracy**: 28.6713286713%
* **Hamlet Accuracy**: 29.8657718121%
* **Romeo Accuracy**: 33.8461538462%
* **Juliet Accuracy**: 43.4343434343%
* **Total Accuracy**: 32.3880597015%

*How does the accuracy obtained compare to the accuracy values obtained in part 1? Why do you think this is the case?*

These accuracy values are noticeably lower for each character (and overall) than in part 1. I think this is the case because as the number of words increases from unigram to bigram, the complexity of the problem increases, meaning more training data is required to accurately model the more specific permutations of words. As a result, each individual permutation of two words is less likely to be seen in the training data and is therefore much less likely to be classified accurately. Having a larger training set (although not possible in this case) would be an effective fix for this problem and would probably yield a greater accuracy than Part 1, if it was given enough data.

**Bonus: Monologue Generation using a predictive keyboard!**

Program5Bigram.py also includes an implementation of a predictive keyboard. As a result, it can be run in the same way specified in Part 2, and the extra credit portion begins after showing the accuracy for each character and the overall accuracy. As this part is an extension of Program5.py, all implementation decisions detailed in Part 1 also apply to this part, but there were a few additional implementation decisions related to the bonus part worth mentioning:

1. 5 options are provided to the user at each step to choose from. After the first word, an additional <eol> option will provide to end the current line and start on the next one.
2. It is assumed that there are at least 5 words to choose from for the first word in the line, and they are taken from the highest frequency in the entire training set for that character.
3. In the event that 5 options cannot be derived from the current word (there weren’t 5 bigrams starting with the current word), the remaining slots will be replaced with the most common words in the entire training set for that character to prevent a line from being stopped early.

When running Program5Bigram.py, the accuracies will be printed for each character as detailed in Part 2. After, a loop will begin for each character, where the user will be tasked with creating 5 lines for each character. For each word of the line, 5 options (or 6 including the <eol> character) will be printed with an index, and the user should enter the index of the option they wish to use to choose that word and proceed. Choosing <eol> will end the line, starting the next one. In the event that that 5th line is finished for a particular character, the final monologue will be printed. After, the process will restart for the next character.

Here are 5-line monologues for generated for each character using the predictive keyboard. Unsurprisingly, they are very dramatic, and I probably had more fun with them than I should have:

**Hamlet:**

1. the king 's brother and the rest shall not think i will come . <eol>
2. i have heard others that would not <eol>
3. and by heaven i will be your most sweet and the very potent poison <eol>
4. of his own honour 'd in this world ! <eol>
5. the rest shall be your only jig or the players cannot live <eol>

**Juliet:**

1. to be married to my romeo ! <eol>
2. i will be gone so thou wilt not yet near . <eol>
3. and yet methinks it was the friar that is my true love . <eol>
4. the shame is my true heart for the lark <eol>
5. i will make confession and my true love 's hand by the world will be gone . <eol>

**Macbeth:**

1. the deed of the worst means . <eol>
2. i will proceed and his house i will have blood <eol>
3. and the time will not yield the way <eol>
4. the door to morrow and his babes . <eol>
5. to all those eyes the worst . <eol>

**Romeo:**

1. i have found him . <eol>
2. the prince i will omit and his wife i will be with <eol>
3. the lark whose house i 'll take . <eol>
4. to make thee better than these poor compounds <eol>
5. i will tear the light upon thy cheeks and in my soul i will be gone ! <eol>

Resources Used:

* <https://stackoverflow.com/questions/268272/getting-key-with-maximum-value-in-dictionary>: Used for getting max key of a dictionary.