Eamon Collins
Ec3bd
HW5

# Naïve Bayes Classifier

## 1.1

I initially misunderstood the instructions and stemmed all the words, using the ‘love’ stems given as examples, but read it again and realized what it was supposed to be. I have a commented out section in the code you can uncomment and it’ll use the standard dictionary, but the way I did it first produced a better classifier so I kept it around. Accuracy results for both are given.

I stemmed all the words as best I could, trying not to deviate from a narrow sense of the meaning of the word. For example, ‘wonderful’ was easily stemmed to ‘wonderfully’, but I didn’t feel that ‘still’ could be translated to ‘stills’ or ‘stilled’ without a significant shift in its meaning.

## 1.3

```
thetaPos = [[ 9.90494196e-01   1.17776504e-03   3.59528274e-04   1.23562016e-03
   1.20256009e-03   1.21909012e-04   8.26501780e-04   3.71925801e-04
   5.82683755e-04   1.71499119e-04   6.81863968e-05   6.61201424e-05
   1.48770320e-04   2.28114491e-03   9.21549484e-04]]
```

**<u>thetaPos</u>**

| | |
|---|---|
| love: | 1.17776504e-03 |
| wonderful: | 3.59528274e-04 |
| best: | 1.23562016e-03 |
| great: | 1.20256009e-03 |
| superb: | 1.21909012e-04 |
| still: | 8.26501780e-04 |
| beautiful: | 3.71925801e-04 |
| bad: | 5.82683755e-04 |
| worst: | 1.71499119e-04 |
| stupid: | 6.81863968e-05 |
| waste: | 6.61201424e-05 |
| boring: | 1.48770320e-04 |
| ?: | 2.28114491e-03 |
| !: | 9.21549484e-04 |
| UNK: | 9.90494196e-01 |

```
thetaNeg = [[ 9.87502330e-01   9.03027574e-04   1.11141855e-04   8.35879369e-04
```

```
  6.32119302e-04   3.93627404e-05   7.82623897e-04   1.69028238e-04
  1.72964512e-03   7.27052970e-04   3.61211029e-04   3.98258315e-04
  5.58024731e-04   3.53338481e-03   1.75048422e-03]]
```

**thetaNeg**

| | |
|---|---|
| love: | 9.03027574e-04 |
| wonderful: | 1.11141855e-04 |
| best: | 8.35879369e-04 |
| great: | 6.32119302e-04 |
| superb: | 3.93627404e-05 |
| still: | 7.82623897e-04 |
| beautiful: | 1.69028238e-04 |
| bad: | 1.72964512e-03 |
| worst: | 7.27052970e-04 |
| stupid: | 3.61211029e-04 |
| waste: | 3.98258315e-04 |
| boring: | 5.58024731e-04 |
| ?: | 3.53338481e-03 |
| !: | 1.75048422e-03 |
| UNK: | 9.87502330e-01 |

The theta values were calculated by counting all the occurrences of each token in the positive or negative training sets for thetaPos and thetaNeg respectively. One was added to the count for each word in order to prevent an absolute zero probability, and then this was divided by the number of samples for each class plus the number of features (the number of features is to offset the Laplace smoothing +1 in the numerator)

## 1.4

MNBC classification accuracy = 0.7016666666666667

Sklearn MultinomialNB accuracy = 0.701666666667

| | Larger Vocab | Smaller Vocab |
|---|---|---|
| **MNBC Accuracy** | 0.7016666666666667 | 0.6683333333333333 |
| **SKLearn MNB Accuracy** | 0.701666666667 | 0.668333333333 |

To test using MNBC you take each sample one at a time and develop probabilities of that sample occurring given that it belongs to one of the classes, so there should be as many probabilities as classes. To do this, you take each value of the theta and exponentiate it by the number of times the corresponding feature occurs in the sample. Then, to prevent underflow, you take the log of it and sum them all to form the probability for that class. The gist of it is seen in this code snippet:

```
for j in range(numSamps):
    for i in range(numFeats):
        posChance[j] += math.log(pow(thetaPos[i,0], Xtest[j,i]))
        negChance[j] += math.log(pow(thetaNeg[i,0], Xtest[j,i]))
```

Whichever class has the highest computed probability is the one you classify the corresponding sample to.

## 1.5

Directly MNBC testing accuracy = 0.7016666666666667

| | **Larger Vocab** | **Smaller Vocab** |
|---|---|---|
| **Directly Tested MNBC Accuracy** | 0.7016666666666667 | 0.6683333333333333 |

How I tested 'directly' was just going file by file and counting the samples in each file. Since I already had the thetaPos and thetaNeg I just needed each files info separately to test it in the exact same way as described in 1.4.

## 1.6

Starts at UNK, goes regular order from there

thetaPosTrue = [0.9985754985754985, 0.4230769230769231, 0.1752136752136752, 0.5042735042735043, 0.43874643874643876, 0.07264957264957266, 0.37037037037037035, 0.17236467236467237, 0.27635327635327633, 0.10113960113960115, 0.038461538461538464, 0.042735042735042736, 0.0868945868945869, 0.5427350427350427, 0.2777777777777778]

thetaNegTrue = [0.9985754985754985, 0.34615384615384615, 0.06267806267806268, 0.36182336182336183, 0.27492877492877493, 0.024216524216524215, 0.3333333333333333, 0.09401709401709402, 0.5185185185185185, 0.2962962962962963, 0.16809116809116809, 0.21082621082621084, 0.25213675213675213, 0.688034188034188, 0.39886039886039887]

The thetas for the BNBC were calculated in much the same way as for the multinomial version, except instead of counting total occurrences of each token, I counted (for each token) the total occurrences of files that contained at least one example of that token. One was added for Laplace smoothing, then the total was divided by the number of samples of the corresponding class plus two.

| | **Larger Vocab** | **Smaller Vocab** |
|---|---|---|
| **BNBC Accuracy** | 0.6933333333333334 | 0.68 |

To make predictions using the BNBC, you check each sample for the occurrence of each token. For each token, if it occurs you use the corresponding value from each of your thetas to determine the chance of that happening given it’s in the class tied to that theta vector. If it doesn’t occur, use the complement of that probability, 1 – theta. Then sum the logs of these feature probabilities for each sample, and classify the sample according to whichever sum is higher.