ECE 448 / CS 440

Fall 2018

MP4

Assignment 4: Application of Naive Bayes

Bo Wang
Yichen Zhou
Weili Liu

Part 1:

Descriptions:

1. The first step we do is to split the train_set into Spam and Ham sets.
2. Then we loop through both subsets and each loop creates a dictionary of seen words as well as their frequencies.
3. For spam dictionary, we go through all the words and calculate P(word | Spam) by (count(word) + alpha)/(n + alpha*(V + 1)), alpha is the laplace smoothing parameter, n is the number of all occurences in the dictionary(sum of values) and V is the length of dictionary. We put those words and their conditional probabilities into a new dictionary. We do the same for the ham dictionary.
4. We loop through the development set and extract each email. For each email, we loop through every word inside. We need to record the sum of log of the probabilities for both the Spam and Ham datasets. If the word is not found in the previous dictionary of conditional probabilities, we compute the log of P(unseen_word | Spam or Ham) by alpha/(n + alpha * (V + 1)). When we finished looping the email, we sum up the log values for both Spam and Ham. If the sum of Spam is larger, we label this email as 0. Otherwise, we label it 1.

Output:

```
zyc@zyc-ubuntu:~/CS440/mp4-code$ python3 mp4.py --training mp4_data/training --d
evelopment mp4_data/development --laplace 0.002
Accuracy: 0.98
F1-Score: 0.9799196787148594
Precision: 0.9838709677419355
Recall: 0.976
```

Laplace smoothing parameter: 0.002

Part 2:

```
zyc@zyc-ubuntu:~/CS440/mp4-code$ python3 mp4.py --training mp4_data/training --d
evelopment mp4_data/development --laplace 0.002 --stemming
Accuracy: 0.968
F1-Score: 0.9682539682539683
Precision: 0.9606299212598425
Recall: 0.976
```

The accuracy, F1-score and precision are lower. This is because the stemming makes some of the words go to the very basic word, ignoring the ‘ing’ or ‘s’. This leads to making different words (ex. Working, worked, works) all become the same word in the dictionary, which ignores their particularity. This would cause poorer performance.

Q&A:

Q1: We have said that this algorithm is called "naive" Bayes. What exactly is so naive about it?

Answer: Because Naive Bayes algorithms makes assumption that may or may not be true. It assumes that each feature of a class is independent of the other features. However, this assumption may not be true.

Q2: Naive Bayes can classify spam quite nicely, but can you imagine classification problems for which naive Bayes performs poorly? Give an example, and explain why naive Bayes may perform poorly.

Answer: When the training set and the development set are very different in their elements, the model would behave very poorly. For instance, if the test set we used contains mostly words not seen in the training set, then the log likelihoods of these worlds would just be the same and do not reflect the actual model. This might cause the the classifier to behave very poorly. In addition, if the features of the dataset are not actually independent of each other, the model would also perform very poorly because the "naive" assumption we made for this classifier does not hold. For example, one word can mean different things when combined with different words, this shows characteristics of dependency of features, which will result in a poor model.

Extra Credit:

We considered viewing two consecutive words as a whole this time, aside from the unigram. We did the same for those bigram sets. The only difference is that the loop number we use is length of dictionary minus 1 because we only need to go to the second last word. When we are calculating the maximum likelihood, we use lambda as the weight for bigram and 1-lambda as the weight for unigram.

The following choices made and optimal performance are using the previous optimal laplace smoothing parameter 0.002

Best parameter: lambda = 0.05

Results:
Bigram Model: lambda = 1

```
zyc@zyc-ubuntu:~/CS440/mp4-code$ python3 mp4.py --training mp4_data/training --d
evelopment mp4_data/development --laplace 0.002
Accuracy: 0.972
F1-Score: 0.9718875502008033
Precision: 0.9758064516129032
Recall: 0.968
```

Optimal Mixture Model: lambda = 0.05

```
zyc@zyc-ubuntu:~/CS440/mp4-code$ python3 mp4.py --training mp4_data/training --d
evelopment mp4_data/development --laplace 0.002
Accuracy: 0.98
F1-Score: 0.9799196787148594
Precision: 0.9838709677419355
Recall: 0.976
```

Q1: Running naive Bayes on the bigram model relaxes the naive assumption of the model a bit. However, is this always a good thing? Why or why not?

Answer: It would not not always be a good thing. The bigram model could make up for the potential mistakes of unigram model since the meaning of words in the email may have relations with each other. However, during testing, bigram may also increase the possibility of checking unseen strings, which, in our model, would reduce the impact of known words. So the output of naive Bayes on the bigram model would depends on our data set.

Q2: What would happen if we did an *N*-gram model where *N* was a really large number?

Answer: It means we are checking a really large set of consecutive words. However, it is very unlikely that we encounter the same consecutive set of words in the test set. So most likely, we would get poorer prediction.