Assignment 5: Naive Bayes Classifiers

11-411/611 Natural Language Processing

Due: Oct 15th, 2020

Introduction

In this assignment, you will create a Naive Bayes text classifier. Your classifier will distinguish speeches given by Republicans (Red) and Democrats (Blue) running for president. (For more information of Naive Bayes Classification, check out Chapter 4.1 & 4.2 in https://web.stanford.edu/~jurafsky/slp3/)

Data

The training dataset includes 46 speeches distributed among 4 candidates, 2 from each party. The test dataset includes 18 different speeches from 3 of the 4 candidates.
The training and testing files have one document (speech) per line. The class of the document is given first, followed by a tab character, followed by the speech (each word is separated by whitespace).
(Note: `test.txt` and `test2.txt` are more or less the same except that `test2.txt` has more unknown words since fewer words appear in `train2.txt`.)

Task1: Programming [50 points]

Subtask 1: Implementation [40 points]

In your file `naivebayes.py`, create a Naive Bayes classifier using the provided template.

1. For each speech, Naive Bayes classifier returns the class $\hat{c}$ which has the maximum posterior probability. By applying Bayes' rule we can get:

$$\hat{c} = \underset{c \in C}{argmax} P(c|speech) = \underset{c \in C}{argmax} \frac{P(speech|c)P(c)}{P(speech)}$$

   We can drop the denominator $P(speech)$.

2. Prior and Likelihood: the Naive Bayes model assumes that all words in a speech are independent of one another given the class. We can write the

likelihood of a speech as:

$$P(f_1, ..., f_n|c) = \prod_{i=1}^{n} P(f_i|c)$$

where f_i is the i^{th} term in the speech and c is the class of speech. You will need to write program to estimate the multinomial distributions P(term—class) and the prior P(c) from the training dataset.

3. Use add-one smoothing when estimating probability. Make sure to add the size of the vocabulary (number of unique words in all classes) to the denominator when normalizing.

4. Probability calculations are done in log space, to avoid underflow and increase speed.

5. Split on whitespace to tokenize.

6. You do not need to deal with the problem of new words in the test data since some of the singletons (words appearing only once) in the training data have been replaced by the token UNKNOWNWORD.

Subtask 2: Evaluation [10 points]

Implement the function `testModel` that tests your classifier and reports the following:

- Overall accuracy (proportion correct) across categories.
- Precision for each category.

$$Precision = \frac{True\,Positives}{True\,Positives + False\,Positives}$$

- Recall for each category.

$$Recall = \frac{True\,Positives}{True\,Positives + False\,Negatives}$$

To evaluate your implementation, train a Naive Bayes classifier using the training data `train.txt`. Use the test file `test.txt` to evaluate your classifier. The code can be run using the command

```
python3 naivebayes.py train.txt test.txt
```

Now repeat your tests using a different pair of training and test data, `train2.txt` and `test2.txt`.

```
python3 naivebayes.py train2.txt test2.txt
```

For reference, our model solution has an accuracy of > 0.90 for the first dataset and > 0.75 for the second dataset.

Task2: Analysis [50 points]

In a new file `task2.txt`, answer the following questions.

1. [15 points] What are some problems with only testing on a test set of 18 speeches? What changes can you think of to make for a better evaluation? Are there drawbacks to your suggested changes?

2. [15 points] Examine the probabilities of the predicted classes for the test set speeches. Are some predictions more "certain" than others? If any of the classifications are wrong, do the probabilities for these indicate relatively low confidence?

3. [20 points] Naive Bayes models can be feature-based; that is, you can then think of each token in the document as consisting of a vector of features. The simplest case is what you have implemented so far: a token's sole feature is the entire word. Additional features might capture aspects of the word's internal structure or context.
   Extend your model to incorporate multiple types of features and code this in `extended.py`. Did classification accuracy improve? Why do you think you saw the effect you did? Be sure to explain the features you tried and measure their impact.

Submission Guidelines

Please submit a zip archive named `handin.zip` containing the following items. Please do not put them in a folder inside the archive.

1. A file called `naivebayes.py` which implements your classifier and reports its accuracy, precision, and recall.

2. `task2.txt`, with brief answers and comments

3. `extended.py` which implements your extended classifier for Task 2 Q3.

4. A file called `README`. Whether or not you collaborated with other individuals or employed outside resources, you must include a section in your `README` documenting your consultation (or non-consultation) of other persons and resources. If you have any additional information to give us, you should also put it there.