Homework 2

Author: Nicholas M. Synovic

Table of Contents

About

The homework assignment description can be found in hw2.pdf. The `hw2.py` script is the executable code to run to generate results.

The dataset used for this assignment was downloaded from here.

Dependencies

To run this code, you will need:

- `Python 3.10`
- `requests`
- `pandas`
  - These can be installed by running `pip install -r requirements.txt`

How To Run

- `python3.10 hw2.py`

Methodology

My methodology follows the algorithm description from Figure 4.2 of the text, I first computed the overall document frequency for each class.

Data Preprocessing

Stop words **were not removed from datasets** and all documents were made lowercase. Additionally, all non alphabetical charachters were removed from the dataset prior to usage.

Data Splitting

The dataset was split with the following method:

1. Load in the positive dataset
2. Use *the first 70%* of indexes to create **the positive training dataset**
3. Use *the next 15%* of indexes to create **the positive development dataset**
4. Use *the remaining 15%* of index to create **the positive testing dataset**
5. Repeat steps 1 - 4 and substitute *the positive dataset* for *the negative dataset* (thus resulting *in the same splits percentages* for **the negative training, development, and testing datasets**)

The results of the splitting are the following:

```
Positive Training Doc. Size     : 3732   (69.9925% of Positive Docs)
Positive Development Doc. Size  : 800    (15.00375% of Positive Docs)
Positive Testing Doc. Size      : 799    (14.985% of Positive Docs)

Negative Training Doc. Size     : 3732   (69.9925% of Negative Docs)
Negative Development Doc. Size  : 800    (15.00375% of Negative Docs)
Negative Testing Doc. Size      : 799    (14.985% of Negative Docs)
```

Naive Bayes Implementation

> **NOTE**: Any `log` operations where done using Python's `math.log10` function
>
> **NOTE**: This algorithm was first tested on the development datasets. Results for these tests can be found in Results

Following the algorithm description from Figure 4.2 of the text, I first computed the overall document frequency for each class.

This was done by performing the following algorithm:

```python
data: List[str] = positiveData + negativeData

positiveClassLog = abs(log10(len(positiveData) / len(data)))
negativeClassLog = abs(log10(len(negativeData) / len(data)))
```

Where `positiveData` and `negativeData` are the **positive dataset** and **negative dataset** calculated from during the data splitting operation respectfully.

Generating Vocabularly

The vocabulary was generated by first joining the positive and negative documents from the training datasets. And then splitting the documents into words by splitting on the spaces in between each word.

Computing Word Frequency

Word frequency is the count of a word within a set of documents in a class. To compute this, I used a `defaultdict(int)` to store the mapping of `{word : count}`.

I first split each document in a class by spaces and joined all the resulting lists of words. I then iterated over each word, appended it to the the list, and set the value for each word to be `1+` the previous value.

Computing Class Likelihoods per Word

The class likelihood per word is the measurement of how likely a word is to appear within a set of documents given a class. It is measured as a percentage.

To compute this, I took the word counts for a given class from Computing Word Frequency and divded is by total number of words within said class. Laplace smoothing of 1 was applied to both the total number of words as well as the word count. The `log10()` was taken of the quotient.

This operation was done twice; once for the positive class and once for the negative class.

The two quotients were then appended to a list and stored in a dictionary. An example dictionary is described below:

```
classLikelihood: dict[str, List[float, float]] = {"word": [0.1, 0.2]}
```

Testing Naive Bayes

Combining the above functions together results in a functional Naive Bayes algorithm.

Tests were done using the development dataset while creating this algorithm. The training and development dataset were concatinated together prior to evaluating the testing dataset.

To determine the class probability of a document, each token of the document was looked up in the class likelihood mapping and summed together. Both the positive and negative class likelihoods were consoluted to determine the positive and negative classes respectfully.

If a word in the testing dataset did not exist within the class likelihood mapping, then no value for that word was added (equivalent to a `+ 0` operation).

If the positive class likelihood was greater than the negative class likelihood, then it was reported that it was a positive class. Else, it was reported as a negative class.

Results

The following table includes the accuracy results from my Naive Bayes implementation.

	True Positive	**True Negative**	**False Positive**	**False Negative**
Development Dataset	73.0%	72.75%	27.0%	27.25%
Testing Dataset	72.59074%	77.09637%	27.40926%	22.90363%

The data generated from the development and testing datasets can be found in the following files:

- negativeDevelopment.json
- positiveDevelopment.json
- negativeTest.json
- positiveTest.json

Very Confident Examples

> **NOTE**: Confidence scores were taken by subtracting the negative class probability from the positive class probability if the document was classified as positive or vice versa for documents classified as negative.

The following examples were *very confident* **positive** examples:

- "neither the funniest film that eddie murphy nor robert de niro has ever made showtime is nevertheless efficiently amusing for a good while before it collapses into exactly the kind of buddy cop comedy it set out to lampoon anyway"
  - Confidence score = `4.412481`
- "witty dialog between realistic characters showing honest emotions touching and tender and proves that even in sorrow you can find humor like blended shades of lipstick these components combine into one terrific story with lots of laughs"
  - Confidence score = `5.160191`

The following examples were *very confident* **negative** examples:

- "in addition to sporting one of the worst titles in recent cinematic history ballistic ecks vs sever also features terrible banal dialogue convenient plotting superficial characters and a rather dull unimaginative car chase"
  - Confidence score = `5.748131`
- "the first question to ask about bad company is why anthony hopkins is in it we assume he had a bad run in the market or a costly divorce because there is no earthly reason other than money why this distinguished actor would stoop so low"
  - Confidence score = `7.212231`

The reason why these examples scored so confidently is because:

1. There were many tokens to analyze. The longer the document, the more likely it is for a Naive Bayes classifer to confidently choose a class given the class likelihoods of the tokens.
2. There are many "unique" words to each class contained within the documents. In other words, the words of one document show up frequently with respect to its class, but not often in the other class(es).

Very Unconfident Examples

> **NOTE**: Confidence scores were taken by subtracting the negative class probability from the positive class probability if the document was classified as positive or vice versa for documents classified as negative.

The following were *very unconfident* **positive** examples:

- "love is a little like a chocolate milk moustache"
  - Confidence score = `0.000307`
- "can i admit xxx is as deep as a petri dish and as as a telephone book but still say it was a guilty pleasure"
  - Confidence score = `0.002818`

The following were *very unconfident* **negative** examples:

- "insufferably naive"
  - Confidence score = `0.0`
- "slummer"
- Confidence scorre = `0.0`

1. There were not many tokens to analyze. The shorter the document, the less likely it is for a Naive Bayes classifer to confidently choose a class given the class likelihoods of the tokens.
2. There are few "unique" words to each class contained within the documents. In other words, the words of one document show up frequently within all classes.
3. The negative examples contained words that were equally represented within the training dataset for all classes, and therefore defaulted to being

negative by the rules of the classifier.

Most Important Features

The top 10 *most important* **positive** features and their weights were:

1. 'with', -2.0226690276635537
2. 'it', -1.934427855005099
3. 'that', -1.8767862868135838
4. 'in', -1.8506017039117106
5. 'is', -1.739451251789044
6. 'to', -1.6794148625966396
7. 'of', -1.4531239565732608
8. 'and', -1.4240558360303712
9. 'a', -1.3871693868896688
10. 'the', -1.2721779107418376

The top 10 *most important* **negative** features and their weights were:

1. 'as', -2.0074523284003867
2. 'it', -1.9017238503451415
3. 'that', -1.883349577578428
4. 'in', -1.8577596121191151
5. 'is', -1.752489260062507
6. 'to', -1.609512319728349
7. 'and', -1.5502694036229117
8. 'of', -1.5307024242650458
9. 'a', -1.430314188584255
10. 'the', -1.267697468931049)]

These are the most important features respectfully because their class likelihoods were the largest out of all the types for either class. However, these as these features are mainly stop words, there is a case to be made that these specifc fetures are not that important as they don't capture the sentiment of a class.