The implementation of a Naïve Bayes Classifier to classify movie reviews into categories from scratch.

Names: Loyd Flores
Department: Computer Science

Queens College, City University of New York
New York, NY

November 20th, 2024

Abstract

This exploration aims to implement the Naive Bayes Algorithm from scratch in Python. The model will be trained on 50,000 movie review files and will be tested on 50,000 more movie files.

Contents

1 Naïve Bayes

1.1 Bag-of-words-features

Bag-of-words (**BoW**) is a statistical language model used to analyze text and documents based on the frequencies of words. Unlike N-Gram Language models that make the *Markov's Independence Assumption* which assumes that each word in a sequence depends only on the previous n-1 words, **BoW** does not account for word order within a document.

For example, given 1 text document containing the string: "I always like foreign films," the Bag-of-words features will look like the following:

Token	Value
"I"	: 1,
"always"	: 1,
"like"	: 1,
"foreign"	: 1,
"films"	: 1

1.2 Prior Probabilities

The Naive Bayes classifier only has two parameters: Prior Probabilities and Feature Probabilities. Prior probabilities are the probabilities of a class. For example, we are tasked to make a classifier that can classify movie reviews into movie categories *comedy and action.* To understand the pattern of what makes a category unique, we are given five movie reviews: three reviews are written for action films and 2 reviews are written for comedy films. The prior probability for **action** is $\frac{3}{5}$ or **60%** and **comedy** $\frac{2}{5}$ or **40%**.

1.3 Feature Probabilities and Laplace Smoothing

Feature Probabilities are the second parameter of the Naive Bayes classifier. Feature probability is the probability of a token appearing in a class. Going back to our movie review example, what is the probability of the word *laughter* appearing given that the movie review is written for a comedy film.

It is safe to assume that the word *laughter* has a higher probability of appearing in a movie review for a comedy film rather than an action film that may contain words with a more serious tone.

The way we get the Feature probability is by counting the number of times word w_i appears divided by the total number of tokens in that class.

$$\text{Feature Probability} = \frac{c(w_i)}{\sum_{W \in \text{class}} W}$$

This is a great way to estimate probabilities for words that appear but fails to generalize on words the model has never seen before because it will assign 0 to the numerator of the fraction which will lead to a zero probability, which is wrong.

To understand why this is wrong, in the five movie reviews that were previously handed to us, it is very unlikely for it to contain all the words usable to write a review for both action and comedy films. It is definitely wrong to assume that because the total number of combinations of words to write reviews for a comedy is probably larger than the words available that are associated with comedy reviews. It also changes through time, which is why it is impossible to have enough data in this context.

Laplace Smoothing ensures that unseen words are accounted for in the probability distribution by adding one to the numerator of each word's probability calculation, thus preventing any probability from becoming zero. To maintain a valid probability distribution, the denominator is adjusted by adding the total size of the vocabulary.

$$\text{Feature Probability with Laplace Smoothing} = \frac{c(w_i) + 1}{\sum_{W \in \text{class}} W + V}$$

1.4 Classifier

Classifying text using the Naive Bayes Classifier involves computing the probability of a string belonging to different classes and selecting the one with the higher probability. To calculate the probability of a string within a specific class, we multiply the prior probability by the probabilities of each token given that class.

Prior Probabilities

$$P(\text{positive}) = 0.4$$
$$P(\text{negative}) = 0.6$$

Feature Probabilities

Word	P(pos)	P(neg)
I	0.09	0.16
always	0.07	0.06
like	0.29	0.06
foreign	0.04	0.15
films	0.08	0.11

Probability Calculation

What is $P(\text{"I always like foreign films"})$?

$$P(\text{"I always like foreign films"}) = \arg\max \begin{pmatrix} P(\text{positive}) \times P(\text{I} \mid \text{positive}) \times P(\text{always} \mid \text{positive}) \times P(\text{like} \mid \text{positive}) \times \\ P(\text{foreign} \mid \text{positive}) \times P(\text{films} \mid \text{positive}), \\ P(\text{negative}) \times P(\text{I} \mid \text{negative}) \times P(\text{always} \mid \text{negative}) \times P(\text{like} \mid \text{negative}) \times \\ P(\text{foreign} \mid \text{negative}) \times P(\text{films} \mid \text{negative}) \end{pmatrix}$$

$$= \arg\max \left(0.4 \times 0.09 \times 0.07 \times 0.29 \times 0.04 \times 0.08,\ 0.6 \times 0.16 \times 0.06 \times 0.06 \times 0.15 \times 0.11 \right)$$

$$= \arg\max \left(0.00000233 \text{ or } 2.33 \times 10^{-6},\ 0.0000057 \text{ or } 5.7 \times 10^{-6} \right)$$

$$P(\text{"I always like foreign films"}) = \text{Negative}$$

2 Implementation of Naive Bayes from Scratch

2.1 Naive Bayes Method

In my implementation of Naive Bayes I split it up into three components. **NB.py** which holds all the main algorithm steps of Naive Bayes. **methods.py** a library that holds all my helper methods. **main.py** driver code to just input the directories of the training and test data and it will execute the program. The assumptions of this implementation is that The Naive Bayes model will only be predicting two classes. Below are the Naive Bayes Algorithm Steps in my implementation:

1. Log "Entering Naive Bayes".
2. Start recording execution time.
3. Sort test subdirectories for consistent labeling and make code dynamic.
4. Compute prior probabilities for class 1 and class 2.
5. Generate Bag-of-Words (BOW) features for each class.
6. Generate mapping for test classes.
7. Compute conditional probabilities with Laplace smoothing (add 1) and extract vocabulary size.
8. Generate predictions on the test set.
9. Save model parameters.

2.2 Other Helper Methods

- **compute_prior_probabilities(data_path)**
  - *Description:* Computes the prior probabilities of classes by counting the number of documents in each class within the specified training data directory.
  - *Input:* Path to the training data directory (`data_path`).
  - *Output:* Class names and their respective prior probabilities.
- **compute_conditional_probabilities_with_add1(bow1, bow2)**
  - *Description:* Computes the conditional probabilities for tokens using Laplace (add-1) smoothing based on the given Bag-of-Words (BOW) features for two classes.
  - *Input:* Bag-of-Words for class 1 (`bow1`) and class 2 (`bow2`).
  - *Output:* Conditional probabilities for each class, sizes of the Bag-of-Words for both classes, and the total vocabulary size.
- **get_validation_labels(class1_folder_path, class2_folder_path, output_file)**
  - *Description:* Extracts ground truth labels for the test set and writes them to an output file for validation.
  - *Input:* Paths to class 1 and class 2 folders (`class1_folder_path`, `class2_folder_path`) and the output file path (`output_file`).
  - *Output:* Writes extracted labels to the specified output file.
- **validate_predictions(actual_labels_file, predictions_file)**
  - *Description:* Compares predicted labels with the actual labels and computes the accuracy of predictions.
  - *Input:* Path to the actual labels file (`actual_labels_file`) and path to the predictions file (`predictions_file`).
  - *Output:* Prints the number of correct predictions, total predictions, and the accuracy percentage.
- **ensure_directory_exists(file_path)**
  - *Description:* Ensures that the directory for the specified file path exists; creates it if it does not.
  - *Input:* File path (`file_path`).
  - *Output:* None (creates directory if necessary).

To understand the bigger picture here is a UML Diagram to explain the program execution flow

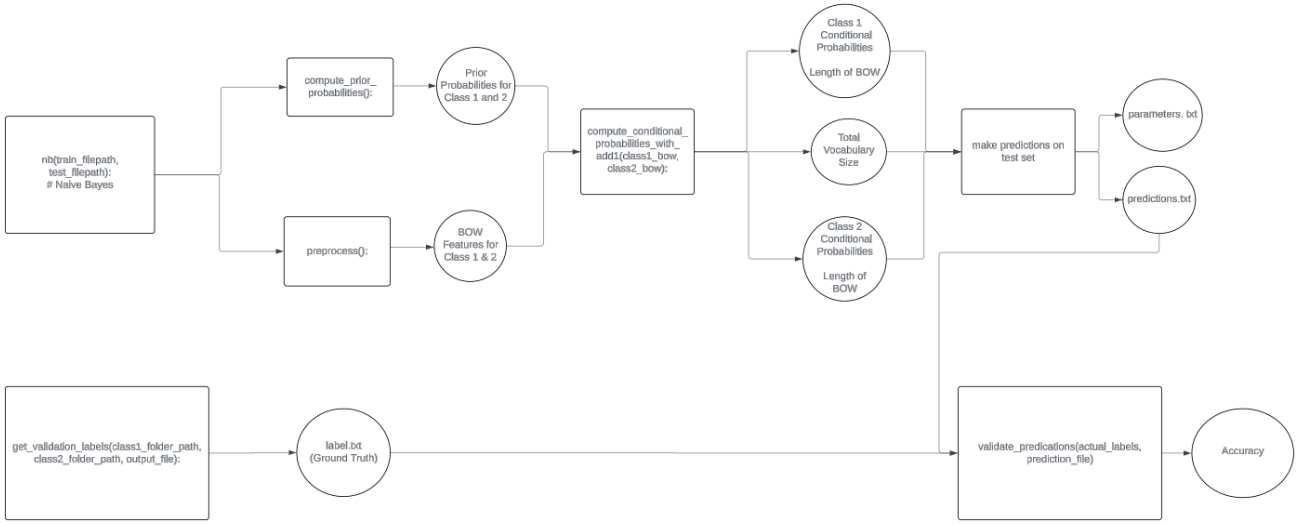

Figure 1: UML Diagram

3 Experiment & Results

3.1 Validation of Implementation

To ensure that our model correctly predicts we will demonstrate its predictive ability on a small test set where we are able to compute alongside the model.

Given the five movie reviews:

Review	**Category**
Fun, Couple, Love, Love	Comedy
Fast, Furious, Shoot	Action
Couple, Fly, Fast, Fun, Fun	Comedy
Furious, Shoot, Shoot, Fun	Action
Fly, Fast, Shoot, Love	Action

The Prior Probabilities are:

$$\text{Action} = \tfrac{3}{5} \text{ or } 60\%, \text{ Comedy} = \tfrac{2}{5} \text{ or } 40\%$$

And the Bag-of-Words features for the two classes are:

BoW for Action Film Reviews	BoW for Comedy Film Reviews
"fast" : 2	"fun" : 3
"furious" : 2	"couple" : 2
"shoot" : 4	"love" : 2
"fun" : 1	"fly" : 1
"fly" : 1	"fast" : 1
"love" : 1	

Action has a total of 11 tokens and Comedy has a total of 9 tokens. In total our vocabulary or $\mathbf{V} = 20$.

To validate our model we will be printing on this movie review: **"fast, couple, shoot, fly"**. To do this we compute the probability of the string to be of both classes and get the higher probability.

$$P(\text{fast, couple, shoot, fly}) = \arg\min \begin{pmatrix} P(\text{Action}) \times P(\text{fast} \mid \text{Action}) \times P(\text{couple} \mid \text{Action}) \times \\ P(\text{shoot} \mid \text{Action}) \times P(\text{fly} \mid \text{Action}), \\ P(\text{Comedy}) \times P(\text{fast} \mid \text{Comedy}) \times P(\text{couple} \mid \text{Comedy}) \times \\ P(\text{shoot} \mid \text{Comedy}) \times P(\text{fly} \mid \text{Comedy}) \end{pmatrix}$$

$$P(\text{fast, couple, shoot, fly}) = \arg\min \begin{pmatrix} 0.60 \times \left(\frac{3}{31}\right) \times \left(\frac{1}{31}\right) \times \left(\frac{5}{31}\right) \times \left(\frac{2}{31}\right), \\ 0.40 \times \left(\frac{1}{29}\right) \times \left(\frac{3}{29}\right) \times \left(\frac{1}{29}\right) \times \left(\frac{2}{29}\right) \end{pmatrix}$$

$$P(\text{fast, couple, shoot, fly}) = \arg\min \left(0.00001949 \text{ or } 1.949 \times 10^{-5}, 0.00000339 \text{ or } 3.39 \times 10^{-6}\right)$$

$$P(\text{fast, couple, shoot, fly}) = \text{Action}$$

To test the same training and testing data set I placed the training data into one directory called training which holds the two categories used to train. To imitate the training data I created a test directory and nested two more folders similar to what I did in the training. I then took one example from each class in train and placed it into their respective classes in test to know for sure if my model is predicting correctly on tranining data. I then added the test review to be predicted **"fast, couple, shoot, fly"** onto both folders to further verify if it is predicting correctly here are the results.

```
Entering Naive Bayes ...
Computing Prior Probabilities...
Computing Prior Probabilities for ./movie-review-small-data/train ...
compute_prior_probabilities() executed in 0.0000 seconds
Prior Probabilities: action 0.6, comedy 0.4

Generating Bag of Words for each class...
Preprocessing files in: ./movie-review-small-data/train\action
preprocessing() executed in 0.0010 seconds
Action BOW: {'fast': 2, 'furious': 2, 'shoot': 4, 'fun': 1, 'fly': 1, 'love': 1}
Preprocessing files in: ./movie-review-small-data/train\comedy
preprocessing() executed in 0.0000 seconds
Comedy BOW: {'fun': 3, 'couple': 2, 'love': 2, 'fly': 1, 'fast': 1}

Class Mapping:
1 -> action
0 -> comedy

Computing Conditional Probabilities with Add-1 Smoothing...
Computing conditional probabilities and smoothing ...
compute_conditional_probabilities_with_add1() executed in 0.0000 seconds

Size of Bag of Word Features for Action: 11
Size of Bag of Word Features for Comedy: 9
Vocabulary Size: 20
NB() executed in 0.0065 seconds
get_validation_labels() executed in 0.0000 seconds

Validation Completed: 3/4 correct predictions.
Accuracy: 75.00%
validate_predictions() in 0.0020 seconds
```

Our model had a 75% accuracy which was predictable as we nested the the review of interest in both action and comedy folder. It predicted action in the the review regardless of where it was nested which validates that our model fully functional.

3.2 Experiment

Given that the definition of Naive Bayes Classifier was correctly implemented we can begin the experiment. The goal is to train from a test data set containing 50,000 text files that contains **6,056,873** tokens and is capable of making predictions on a test data set of 50,000 text files.

3.3 Model Predictions and Validation

Here are the results

Output:

```
$ python -u "c:\Users\usflo\OneDrive\desktop\fall24\csci366\hw2\main.py"
Entering Naive Bayes ...

Computing Prior Probabilities...
Computing Prior Probabilities for ../../../../../Downloads/nlp-hw2/movie-review-HW2/aclImdb/train ...
compute_prior_probabilities() executed in 0.0240 seconds
Prior Probabilities: neg 0.5, pos 0.5

Generating Bag of Words for each class...
Preprocessing files in: ../../../../../Downloads/nlp-hw2/movie-review-HW2/aclImdb/train\neg
preprocessing() executed in 59.5376 seconds

Preprocessing files in: ../../../../../Downloads/nlp-hw2/movie-review-HW2/aclImdb/train\pos
preprocessing() executed in 60.4219 seconds

Class Mapping:
1 -> neg
0 -> pos

Computing Conditional Probabilities with Add-1 Smoothing...
Computing conditional probabilities and smoothing ...
compute_conditional_probabilities_with_add1() executed in 0.0320 seconds

Size of Bag of Word Features for Neg: 2996321
Size of Bag of Word Features for Pos: 3060552
Vocabulary Size: 74891
NB() executed in 253.2344 seconds
get_validation_labels() executed in 0.5640 seconds

Validation Completed: 16784/25000 correct predictions.
Accuracy: 67.14%
validate_predictions() in 0.0470 seconds
```

3.4 Conclusion

I learned that it was pretty simple to implement Naive Bayes from scratch. The challenge was fine tuning the model. On a small scale it is quite easy to calculate probabilities by hand but given such a large corpus to train and predict on it was a little harder to debug and understand what the model and it felt like a black box that just spat out probabilities. I also noticed that in my implementation of Naive bayes it would get a lot of wrong predictions because it did not really pick up context that well.
For example in this entry:
Correct label : Positive
Assigned Label = Negative

review:
"this is one of the best movies have ever seen br br it so full of details and every time you see it you ll find new things like then the father is in the shower but still only hears one voice and when the girls flute they can do it at the same time cause then there would be two girls and there aren br br have some problem finding out about in the middle of the movie their uncle visit them but why does his wife freak out else fantastic movie the best asian movie ever br br hope people will enjoy it there have been so many movie where the main character is skit so the machinist secret window and so on but this movie is way better than them"
As you can see there are aa lot of positive words like 'best, full of details' but it these are like redeeming qualities that saved the movie and the model did not understand that the use of these words doesn't mean the review is good