Aakash Mahindreker · a few seconds ago · 3 min read

Naive Bayes Implementation: Text Classification

INTRODUCTION:

Naive Bayes is the classification method used as a supervised machine learning model for classification of texts. It uses probabilities and conditional probabilities to classify the text. In this blog we are using the dataset for Rotten Tomatoes Review. https://www.kaggle.com/datasets/ulrikthygepedersen/rotten-tomatoes-reviews

In the data we have two columns: Freshness and Review. The Freshness are of two type, "fresh" and "rotten". We consider fresh as positive and rotten as negative type of review. We build a classification model to classify whether the review is positive or negative i.e "fresh" or "rotten" using Naive Bayes.

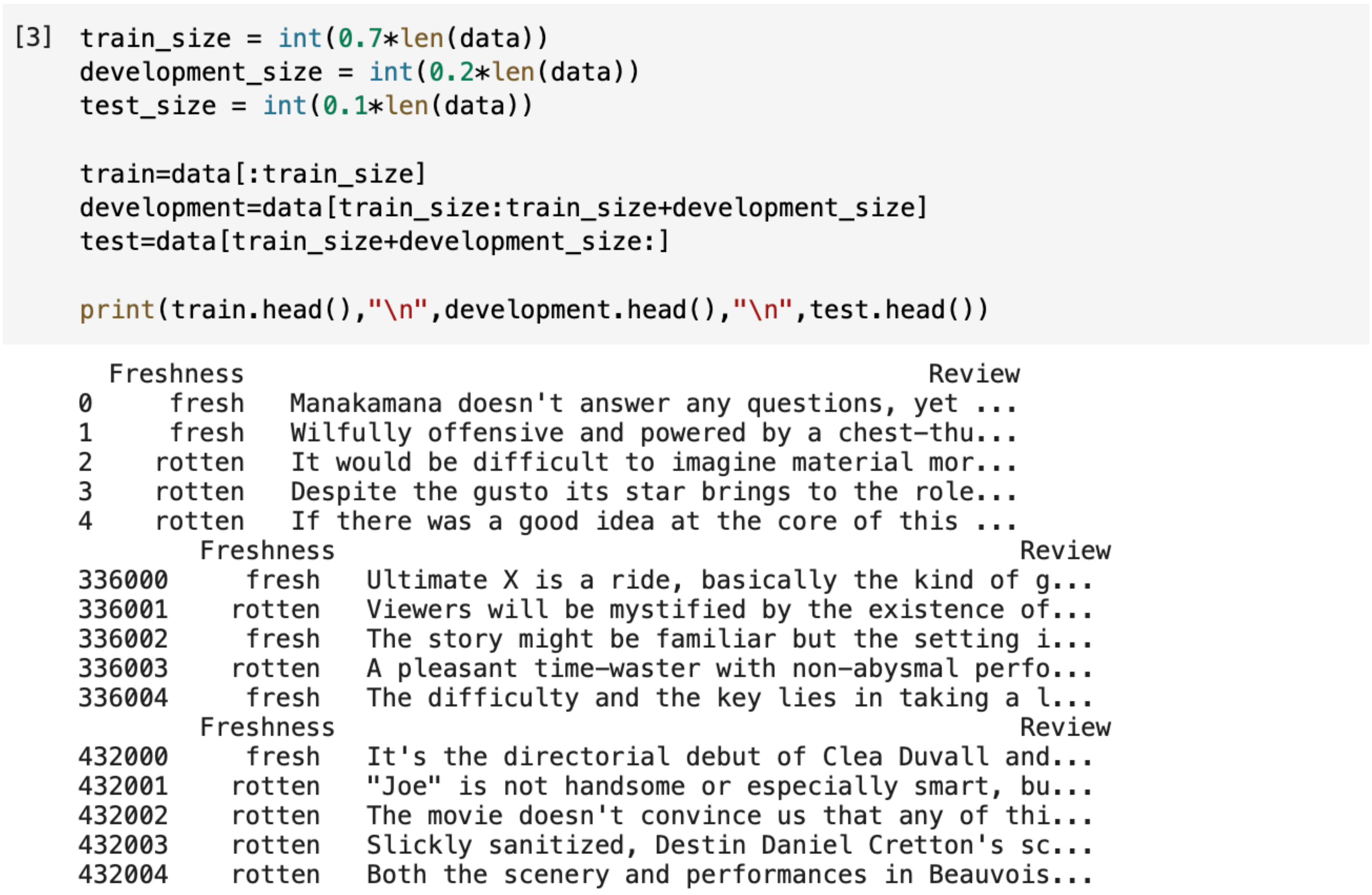
PROCEDURE:

The following is the process that is followed for the implementation of Naïve Bayes Classifier:

i) Split the data into Train, development and test data:

Split the data as shown below:

Split Data



```
[3] train_size = int(0.7*len(data))
    development_size = int(0.2*len(data))
    test_size = int(0.1*len(data))

    train=data[:train_size]
    development=data[train_size:train_size+development_size]
    test=data[train_size+development_size:]

    print(train.head(),"\n",development.head(),"\n",test.head())
```

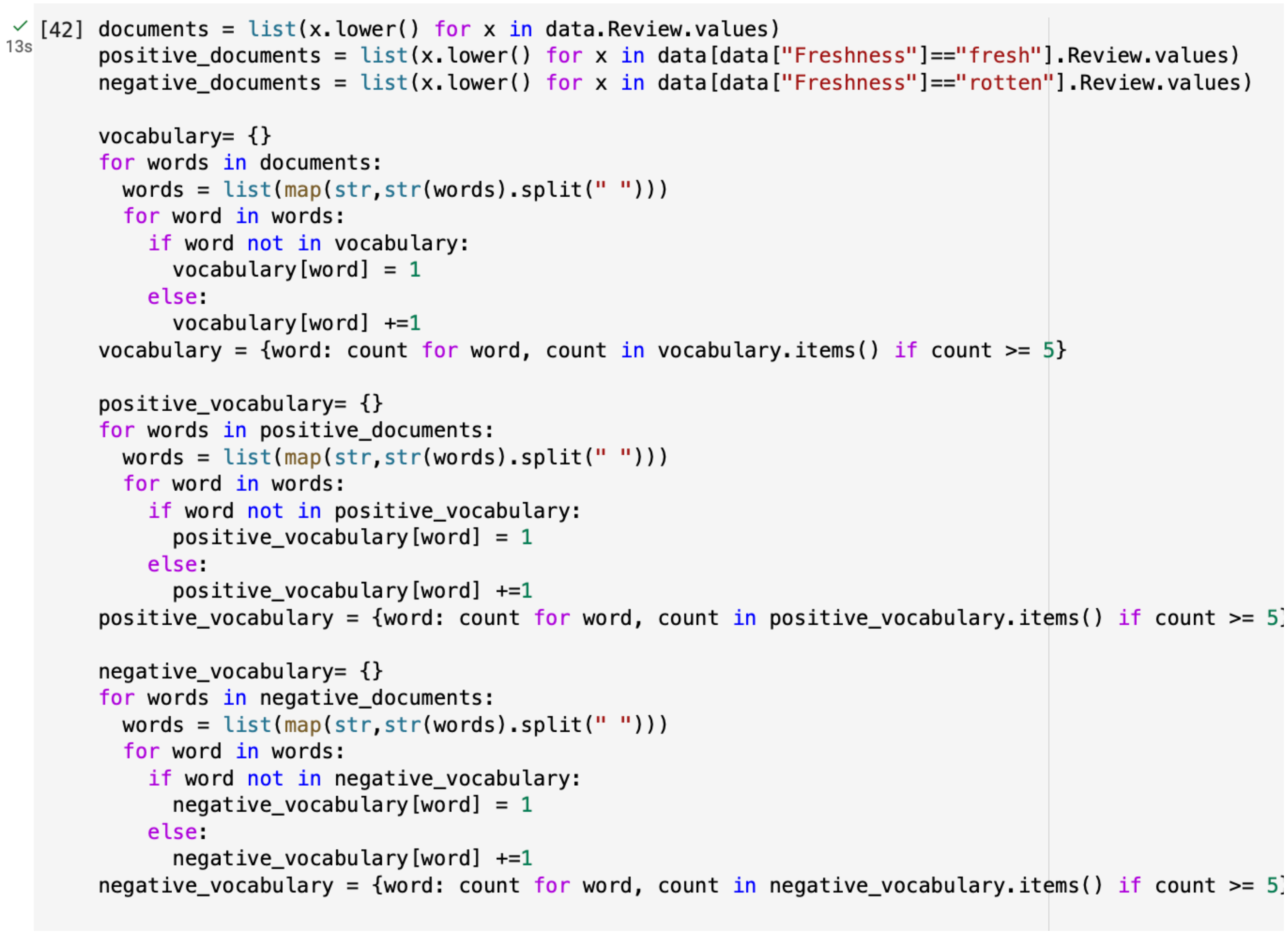
```
  Freshness                                             Review
0     fresh   Manakamana doesn't answer any questions, yet ...
1     fresh   Wilfully offensive and powered by a chest-thu...
2    rotten   It would be difficult to imagine material mor...
3    rotten   Despite the gusto its star brings to the role...
4    rotten   If there was a good idea at the core of this ...
        Freshness                                             Review
336000     fresh   Ultimate X is a ride, basically the kind of g...
336001    rotten   Viewers will be mystified by the existence of...
336002     fresh   The story might be familiar but the setting i...
336003    rotten   A pleasant time-waster with non-abysmal perfo...
336004     fresh   The difficulty and the key lies in taking a l...
        Freshness                                             Review
432000     fresh   It's the directorial debut of Clea Duvall and...
432001    rotten   "Joe" is not handsome or especially smart, bu...
432002    rotten   The movie doesn't convince us that any of thi...
432003    rotten   Slickly sanitized, Destin Daniel Cretton's sc...
432004    rotten   Both the scenery and performances in Beauvois...
```

Here we are considering 70% of the data for training, 20% of data for validation or development and 10% of data for testing.

ii) Build a vocabulary list:

Moving forward, we create a vocabulary list where we store the count of words in the data set. This will make it easier for the calculation of probabilities.

Build Vocabulary List



✓ [42] 13s

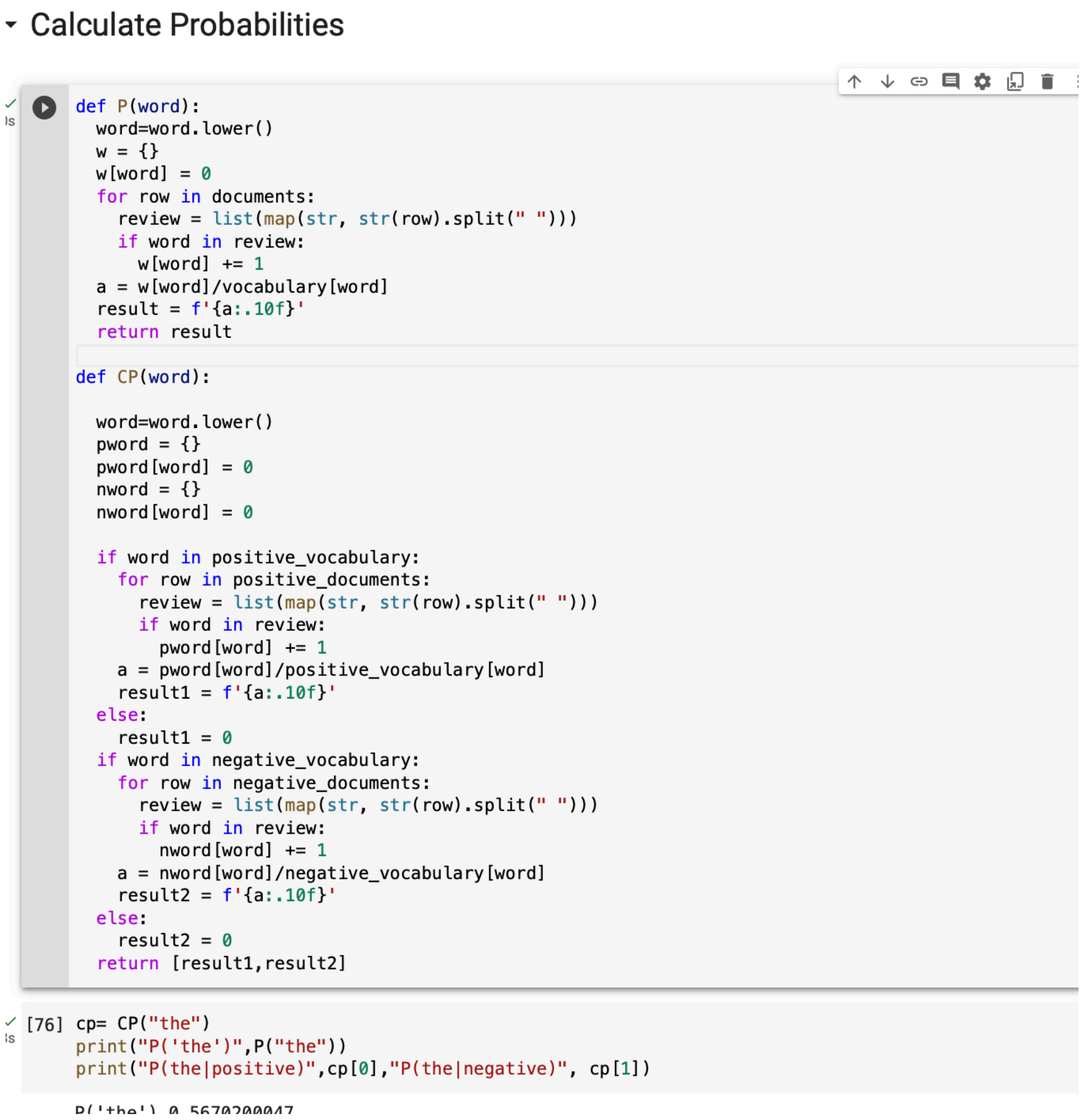
```
documents = list(x.lower() for x in data.Review.values)
positive_documents = list(x.lower() for x in data[data["Freshness"]=="fresh"].Review.values)
negative_documents = list(x.lower() for x in data[data["Freshness"]=="rotten"].Review.values)

vocabulary= {}
for words in documents:
  words = list(map(str,str(words).split(" ")))
  for word in words:
    if word not in vocabulary:
      vocabulary[word] = 1
    else:
      vocabulary[word] +=1
vocabulary = {word: count for word, count in vocabulary.items() if count >= 5}

positive_vocabulary= {}
for words in positive_documents:
  words = list(map(str,str(words).split(" ")))
  for word in words:
    if word not in positive_vocabulary:
      positive_vocabulary[word] = 1
    else:
      positive_vocabulary[word] +=1
positive_vocabulary = {word: count for word, count in positive_vocabulary.items() if count >= 5

negative_vocabulary= {}
for words in negative_documents:
  words = list(map(str,str(words).split(" ")))
  for word in words:
    if word not in negative_vocabulary:
      negative_vocabulary[word] = 1
    else:
      negative_vocabulary[word] +=1
negative_vocabulary = {word: count for word, count in negative_vocabulary.items() if count >= 5
```

We have 3 vocabulary lists, one for the words occurring in the entire data, second for the words occurring in reviews whose freshness is of type fresh, and third is the list of words occurring in the reviews whose freshness is of type rotten, stored as dictionary naming "vocabulary", "positive_vocabulary" and "negative_vocabulary", respectively.

iii) Calculate the probability and conditional probability:

The general formula is used for calculating the probability of occurrence of word. We also calculate the conditional probabilities to know which class the review could be by the probability of its words occurring in the class.

In the below code, function P takes an argument for word which of type string and returns its probability, whereas function CP takes and argument for word and returns the probability of word in "fresh" and probability of the word in "rotten".



Calculate Probabilities

```
def P(word):
  word=word.lower()
  w = {}
  w[word] = 0
  for row in documents:
    review = list(map(str, str(row).split(" ")))
    if word in review:
      w[word] += 1
  a = w[word]/vocabulary[word]
  result = f'{a:.10f}'
  return result

def CP(word):

  word=word.lower()
  pword = {}
  pword[word] = 0
  nword = {}
  nword[word] = 0

  if word in positive_vocabulary:
    for row in positive_documents:
      review = list(map(str, str(row).split(" ")))
      if word in review:
        pword[word] += 1
    a = pword[word]/positive_vocabulary[word]
    result1 = f'{a:.10f}'
  else:
    result1 = 0
  if word in negative_vocabulary:
    for row in negative_documents:
      review = list(map(str, str(row).split(" ")))
      if word in review:
        nword[word] += 1
    a = nword[word]/negative_vocabulary[word]
    result2 = f'{a:.10f}'
  else:
    result2 = 0
  return [result1,result2]
```

```
[76] cp= CP("the")
     print("P('the')",P("the"))
     print("P(the|positive)",cp[0],"P(the|negative)", cp[1])
```

```
P('the') 0.5670200047
```

iv) Calculate accuracy using development data:

Here we create a prediction model and use the probability and conditional probability functions to predict the most probable class of the input review. To check the validation accuracy, we input the reviews from development data and predict its class, either "fresh" or "rotten". If the output of the prediction is same as the actual freshness of the review, then we increase the correctness. Else, we increase the loss. The function along with the output of the sample inputs of size 10 from development data are shown below.

```
✓ [106] def predict_class(review):
0s        rwords = [word.lower() for word in review.split()]
          words = []
          for word in rwords:
            if len(word)>3:
              words.append(word)
          pp = 1
          np = 1

          for word in words:
            if  word in vocabulary and P(word) > str(0.001):
              cp = CP(word)
              pp *= float(cp[0])
              np *= float(cp[1])
              if np<pp:
                return "fresh"
              else:
                return "rotten"
          return "rotten"

        #Calculate accuracy using Development data
        def evaluate(docs, n):
          c=0
          w=0
          accuracy = []
          loss = []
          i=1
          for row in docs.values:
            if(predict_class(row[1])==row[0]):
              c+=1
              accuracy.append(c/i)
            else:
              w+=1
              loss.append(w/i)
            i+=1
            if i==n:
              break
          return [accuracy,loss]
```

```
✓ [107] dev_eval10 = evaluate(development, 10)
35s
```

```
✓ [108] v_acc = sum(dev_eval10[0])/len(dev_eval10[0])*100
0s      print("Validation Accuracy:",v_acc)
```

```
        Validation Accuracy: 51.1111111111111
```

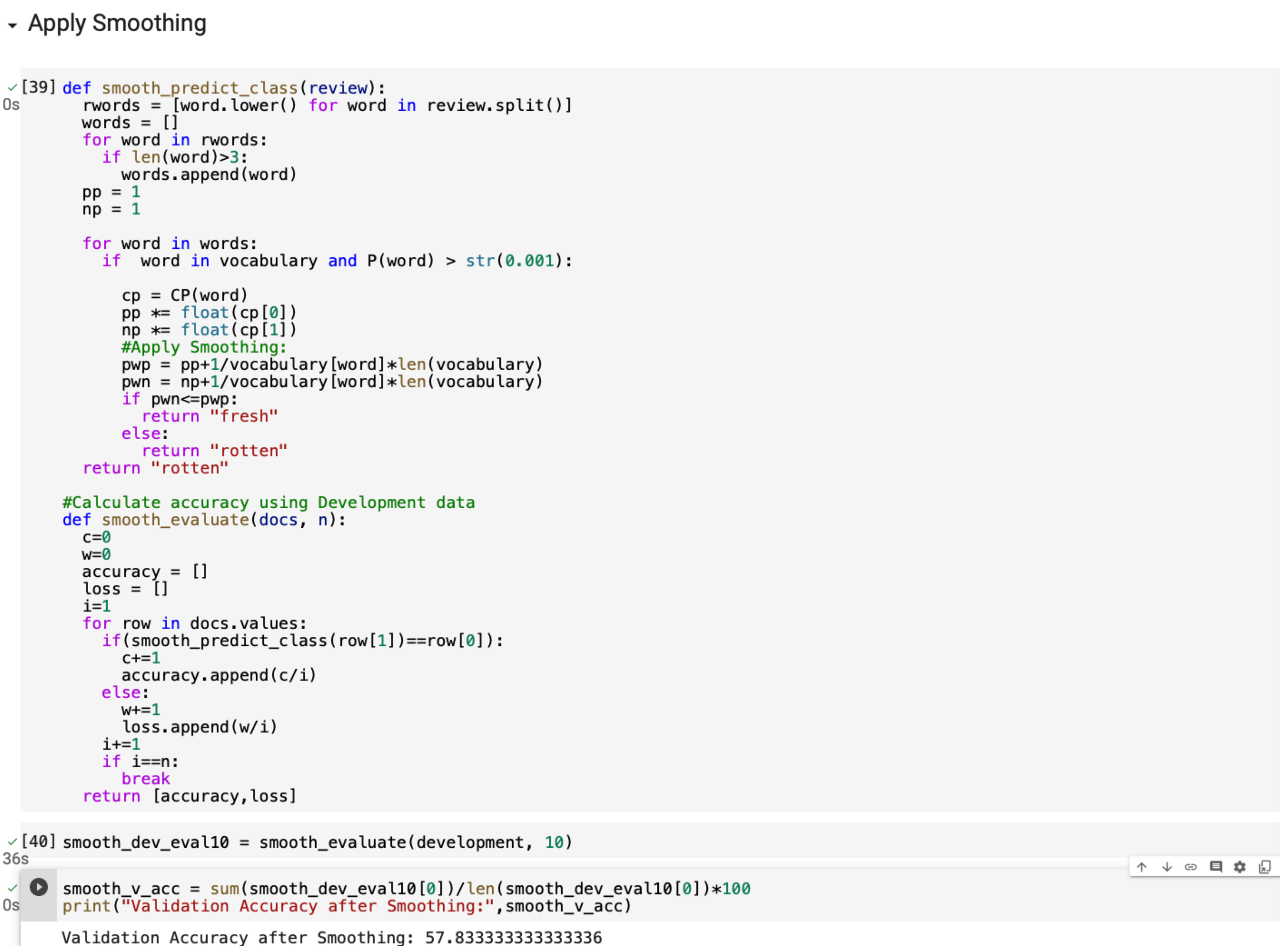
```
✓       v_loss = sum(dev_eval10[1])/len(dev_eval10[1])*100
0s      print("Validation loss:",v_loss)
```

```
        Validation loss: 70.95238095238095
```

v) Experimenting to improve accuracy:

We see that the Validation Accuracy is 51.11% and the loss is 70.95%. Since, this blog is a sample guide for the implementation of Naive Bayes classification, we are not considering the smoothing. We can use Laplace Smoothing which is also known as additive smoothing to improve the accuracy of the classifier. Moreover, the words can be filtered, punctuations can be removed and words can be kept in lower case to improve the accuracy.

Apply Smoothing Sample:



▾ Apply Smoothing

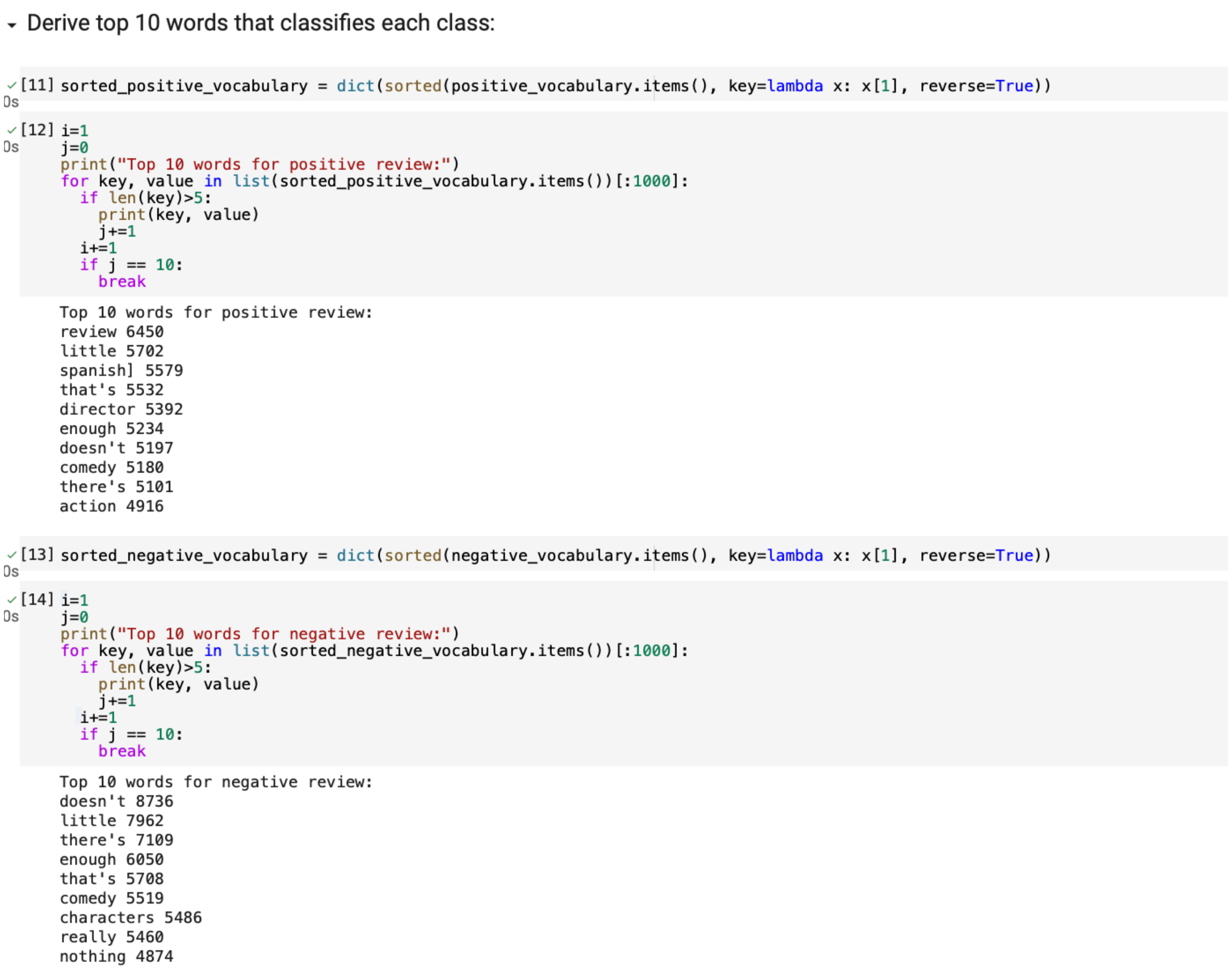
```
✓ [39] def smooth_predict_class(review):
0s       rwords = [word.lower() for word in review.split()]
         words = []
         for word in rwords:
           if len(word)>3:
             words.append(word)
         pp = 1
         np = 1

         for word in words:
           if  word in vocabulary and P(word) > str(0.001):

             cp = CP(word)
             pp *= float(cp[0])
             np *= float(cp[1])
             #Apply Smoothing:
             pwp = pp+1/vocabulary[word]*len(vocabulary)
             pwn = np+1/vocabulary[word]*len(vocabulary)
             if pwn<=pwp:
               return "fresh"
             else:
               return "rotten"
         return "rotten"

       #Calculate accuracy using Development data
       def smooth_evaluate(docs, n):
         c=0
         w=0
         accuracy = []
         loss = []
         i=1
         for row in docs.values:
           if(smooth_predict_class(row[1])==row[0]):
             c+=1
             accuracy.append(c/i)
           else:
             w+=1
             loss.append(w/i)
           i+=1
           if i==n:
             break
         return [accuracy,loss]
```

```
✓ [40] smooth_dev_eval10 = smooth_evaluate(development, 10)
36s
```

```
✓      smooth_v_acc = sum(smooth_dev_eval10[0])/len(smooth_dev_eval10[0])*100
0s     print("Validation Accuracy after Smoothing:",smooth_v_acc)

       Validation Accuracy after Smoothing: 57.833333333333336
```

By Applying smoothing to the prediction function, we were able to increase the validation accuracy to 57.83%.

Deriving Top 10 words of Fresh and Rotten reviews:



▾ Derive top 10 words that classifies each class:

```
[11] sorted_positive_vocabulary = dict(sorted(positive_vocabulary.items(), key=lambda x: x[1], reverse=True))
0s
```

```
[12] i=1
0s   j=0
     print("Top 10 words for positive review:")
     for key, value in list(sorted_positive_vocabulary.items())[:1000]:
       if len(key)>5:
         print(key, value)
         j+=1
       i+=1
       if j == 10:
         break
```

```
Top 10 words for positive review:
review 6450
little 5702
spanish] 5579
that's 5532
director 5392
enough 5234
doesn't 5197
comedy 5180
there's 5101
action 4916
```

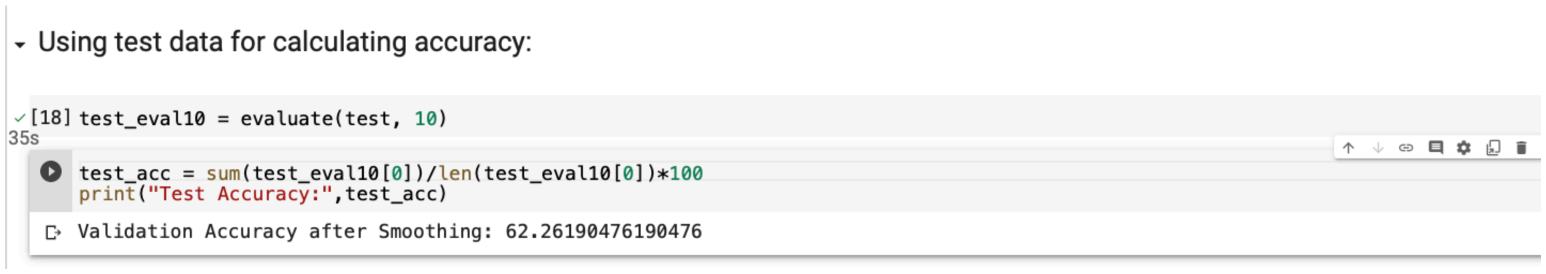
```
[13] sorted_negative_vocabulary = dict(sorted(negative_vocabulary.items(), key=lambda x: x[1], reverse=True))
0s
```

```
[14] i=1
0s   j=0
     print("Top 10 words for negative review:")
     for key, value in list(sorted_negative_vocabulary.items())[:1000]:
       if len(key)>5:
         print(key, value)
         j+=1
       i+=1
       if j == 10:
         break
```

```
Top 10 words for negative review:
doesn't 8736
little 7962
there's 7109
enough 6050
that's 5708
comedy 5519
characters 5486
really 5460
nothing 4874
```

vi) Test Data to calculate the accuracy:

The prediction function was able to generate the accuracy of 62.26% upon the 10 reviews in testing dataset.



▾ Using test data for calculating accuracy:

```
[18] test_eval10 = evaluate(test, 10)
35s
```

```
test_acc = sum(test_eval10[0])/len(test_eval10[0])*100
print("Test Accuracy:",test_acc)
```

```
Validation Accuracy after Smoothing: 62.26190476190476
```

CHALLENGES:

The focus was on creating the Naive Bayes Classifier. However, while developing the classifier, we come across the type of data we have. The data had spelling errors, grammar errors, blank spaces, etc which were enough to reduce the performance of the classifier. These can be avoided by using stop words. The top 10 words that probably classifies the text as fresh and rotten respectively are shown above.

The reviews only in English language can be classified as the dataset does not include the other languages. Even though the dataset is large, it does not have few words to to find the probability, thus the probability becomes 0 entirely.

CONTRIBUTION:

Implemented Naïve Bayes Classifier with Validation Accuracy of 51.11% , 57.83% with smoothing and Testing Accuracy of 62.26%, tried experimenting Laplace Smoothing and identified the top 10 words of the classes "fresh" and "rotten". Derived top 10 words from each class by filtering the stop words in the reviews.

SOURCES:

Dataset used:

rt_reviews.csv
Download CSV • 68.02MB

Source code:

1002027304_DM_ASSIGNMENT.ipynb
Download IPYNB • 25KB

REFERENCES:

1. Ali Hamza, Yousra Javed, and Ghulam Abbas, “A Comparative Study of Decision Tree and Naive Bayes Algorithm for Classification Problems”, International Journal of Emerging Trends & Technology in Computer Science, Vol. 2, No. 2, March-April 2013.
2. Rishabh Misra and Divya Bansal, “A Comparative Study of Naive Bayes Classifier and Decision Trees in Spam Detection”, International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 3, No. 5, May 2013.
3. S. Srinivasan, “Text Classification using Naive Bayes: A Survey”, International Journal of Computer Applications, Vol. 73, No. 19, pp. 29-34, July 2013.
4. Andrew McCallum and Kamal Nigam, “A Comparison of Event Models for Naive Bayes Text Classification”, AAAI/ICML-98 Workshop on Learning for Text Categorization, pp. 41-48, July 1998.