Project #1: Naive Bayes Classifier for Sentiment Analysis

Suhalim Nicolai Christian / 수할림 니콜라이 크리스티안 / 2022313052

Naïve Bayes Classifier (NBC) is a supervised machine learning algorithm that is used for classification tasks such as text classification. There are some assumptions that needed to be made for NBC, those are:

- Feature independence: The features of the data are conditionally independent of each other, given the class label.
- Continuous features are normally distributed: If a feature is continuous, then it is assumed to be normally distributed within each class.
- Discrete features have multinomial distributions: If a feature is discrete, then it is assumed to have a multinomial distribution within each class.
- Features are equally important: All features are assumed to contribute equally to the prediction of the class label.
- No missing data: The data should not contain any missing values.

From these assumptions, in order to calculate the NBC, we use Bayes theorem, which can be formulated as:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

where,

- $P(A)$ is called the prior probability of A, meaning the probability of event before evidence is seen.
- $P(B)$ is called the marginal probability or the probability of evidence.
- $P(A|B)$ is called the posterior probability of B, meaning the probability of event after evidence is seen.
- $P(B|A)$ is called the likelihood probability, meaning the likelihood that a hypothesis will come true based on the evidence.

In this assignment, we were given training dataset and testing dataset in the form of CSV files. These files each consist of stars or the ratings and the text reviews left by the user. For simplicity, if the rating is 5, then the review has a positive sentiment, otherwise it has a negative sentiment. We were required to train NBC model using the training dataset and use this NBC model to predict the sentiment of a review in the testing dataset. With the goal of analyzing the accuracy of an NBC model, we utilize 10%, 30%, 50%, 70%, and 100% of the training dataset to train the NBC model.

In order to complete this assignment, first, we need to pre-process the data given first before eventually train the model. To accomplish this task, we use the "preprocess.py" file. The steps to pre-process the data are by:

1. Read the CSV file.
2. Get the list of stopwords from the "stopwords.txt" file.
3. Convert the text into lowercase.
4. Remove special characters from the text.
5. Remove stopwords from the text.

In this way, we were able to get the processed or cleaned text to be used as our features. As you can see in the Python code, we use *for* loop to implement this pre-processing of the reviews. First, we convert the review into all lowercase (line 65), then, we remove the special characters (line 68). After that, we get the label as positive or negative depending on the star of the review. If the star is equal to 5, it means that it has a positive label, else, negative label (line 72-76). The reason of why we need to calculate the total number of positive and negative sentiments from the train dataset is because this number will be needed to predict the label from the test dataset later on. This number will become $P(A)$ in the Bayes Theorem. We then split the text, and iterate each word. Each word from the review that is not a blank space and the word is not in the stopwords list will be counted and saved in the *word_features* variable (line 82-89). Hence, the *word_features* variable will store the frequency of a word from the whole data. Finally, we saved the pre-processed words or cleaned words as list back towards the original text review. In addition, if we are going to utilized only some part of the dataset, we set a target number of data to be utilized and use a counter to check whether the data utilized is as requested or not (line 94-97). We return the modified data, the number of positive and negative sentiments in the data, as well as the frequency of words from the data.

```
preprocess.py > preprocess
1  '''
2      The file to pre-process the text
3  '''
4
5  import csv
6  import re
7
8  from collections import defaultdict
9
10 # convert the row data from the csv file
11 def convert_row(headers, row):
12     header_item_dictionary = {}
13
14     for header, item in zip(headers, row):
15         header_item_dictionary[header] = item
16
17     return header_item_dictionary
18
19 # get stopwords
20 def get_stopwords() -> list:
21     file_name = "stopwords.txt"
22
23     fh = open(file_name, 'r')
24     lines = fh.readlines()
25     fh.close()
26
27     stopwords = [i.strip() for i in lines]
28
29     return stopwords
30
31 # main driver to pre-process the data
32 def preprocess(file_path, data_use=1) -> tuple[list, dict, dict]:
33     # open and read the csv file
34     fh = open(file_path, "r")
35
36     csv_reader = csv.reader(fh)
37     headers = next(csv_reader)
38
39     data = []
40
41     for row in csv_reader:
42         item_dictionary = convert_row(headers, row)
43         data.append(item_dictionary)
44
45     fh.close()
46
47     stopwords = get_stopwords()
```

```
preprocess.py > preprocess
32 def preprocess(file_path, data_use=1) -> tuple[list, dict, dict]:
49     # variable to store all the word features
50     word_features = defaultdict(int)
51
52     # total data's labels count
53     data_labels_count = {'pos': 0, 'neg': 0}
54
55     data_count = 0
56
57     # number of training/test data to be utilize
58     target_data_count = round(data_use * len(data))
59
60     # pre-process the data
61     for row in data:
62         text = row['text']
63
64         # lowercase the text
65         text = text.lower()
66
67         # remove special chars
68         text = re.sub('[!\"#$%&\'()*+,-./:;<=>?@[\]^_`{|}~]+', ' ', text)
69
70         label = int(row['stars'].strip())
71
72         # if the sentiment is positive sentiment
73         if label == 5:
74             data_labels_count['pos'] += 1
75         else: # negative sentiment
76             data_labels_count['neg'] += 1
77
78         split_text = text.split()
79
80         cleaned_word = []
81
82         for word in split_text:
83             word = word.strip()
84
85             if (word != '') and (word not in stopwords):
86                 word_features[word] += 1
87
88                 # save the pre-process word in the list
89                 cleaned_word.append(word)
90
91         # change the text into the pre-processed words list
92         row['text'] = cleaned_word
93
94         data_count += 1
```

```python
preprocess.py > preprocess
def preprocess(file_path, data_use=1) -> tuple[list, dict, dict]:
        if data_count == target_data_count:
            break

    return (data, data_labels_count, word_features)

```

The second step after the data is pre-processed is the training of the NBC model itself. To do this, we created “training.py” file to implement the training of the NBC model. Again, for simplicity, we are not going to use all of the word features that we have received from the pre-processed data, however, we are only going to use the top 1000 most common words. Meaning is that we are going to select 1000 words that have the highest frequency (line 34). From this top 1000 most common words, we calculate the frequency of each word towards each sentiment. We then saved these words’ frequencies in a dictionary called *train_words_pos* for the words that give a positive sentiment in the review, and *train_words_neg* for the words that give a negative sentiment in the review (40-49). The next step is for us to calculate the likelihood probability ($P(B|A)$) of each word with respect to the sentiment they produce (line 51-53). For example, the probability of the word “food” in the review that gives a positive sentiment is 10 out of 20, then we can rewrite this as $P(B = positive|A = food) = \frac{10}{20}$. The number 20 here means the total number of each word frequencies with respect to each sentiment. In addition, to prevent the problem of zero probability, we apply Laplace smoothing. The way to do Laplace smoothing is as follow:

$$P(B_i = b|A = a) = \frac{N_{B_i=b,A=a} + \alpha}{N_{A=a} + \alpha n_i}$$

where,

- $N_{B_i=b,A=a}$ is the number of training examples for which $B_i = b, A = a$.
- $N_{A=a}$ is the number of examples for which $A = a$.
- n_i is the number of values B_i can take.
- α is the smoothing parameter, in which for Laplace smoothing is equal to 1.

To calculate the likelihood probability, we created a function called *calc_likelihood()* (line 5-15). We run this function twice, since we need to calculate the likelihood probability of both sentiments. For example, the word “food” can give positive sentiment in a review, but it also can give a negative sentiment in another review. For this reason, we calculate both likelihood probability. Inside this function, we calculate the total frequency of the words that give a positive or negative sentiment, and saved it in *total_sample* variable (line 9). We also counted the total number of words in each sentiment and saved it in *total_word_count* variable (line 10). Finally, we can calculate the likelihood probability with Laplace smoothing as written in the formula above, and we return the dictionary of the likelihood probability. After that, we also need to count the prior probability ($P(A)$) of the trained data. We use the *calc_prior_prob()* function written in line 17-29. This function will only calculate the probability of the sentiment to be positive and negative with respect to the total data exist. It will return the dictionary as well, that stores the prior probability of positive and negative sentiment. Lastly, we can return these likelihood probabilities that we have calculated and the prior probabilities as well.

training.py > ...

```
31  # main driver to train the NBC model
32  def train(train_data: list, train_data_count: dict, train_words: dict) -> tuple[dict, dict, dict]:
33      # select top 1000 features
34      top_thousand = dict(sorted(train_words.items(), key=lambda item: item[1], reverse=True)[:1000])
35  
36      # store the frequency of the top 1000 words for positive and negative sentiment
37      train_words_pos = {k: 0 for k in top_thousand.keys()}
38      train_words_neg = {k: 0 for k in top_thousand.keys()}
39  
40      # get the number of frequency of word with respect to the sentiment they gave
41      for row in train_data:
42          label = int(row['stars'].strip())
43  
44          for word in row['text']:
45              if word in top_thousand.keys():
46                  if label == 5:
47                      train_words_pos[word] += 1
48                  else:
49                      train_words_neg[word] += 1
50  
51      # calculate the likelihood probability
52      likelihood_pos = calc_likelihood(train_words_pos)
53      likelihood_neg = calc_likelihood(train_words_neg)
54  
55      # calculate the prior probability
56      prior_prob = calc_prior_prob(train_data_count)
57  
58      # return the model
59      return (likelihood_pos, likelihood_neg, prior_prob)
```

training.py > ...

```python
# function to calculate the likelihood probability
def calc_likelihood(word_features: dict, smoothing=1) -> dict:
    likelihood = {}

    total_sample = sum(word_features.values())
    total_word_count = len(word_features)

    for word, count in word_features.items():
        likelihood[word] = ((count + smoothing) / (total_sample + (smoothing * total_word_count)))

    return likelihood

# function to calculate the prior probability
def calc_prior_prob(data_labels_cnt: dict) -> dict:
    pos_cnt = data_labels_cnt['pos']
    neg_cnt = data_labels_cnt['neg']

    total_data = pos_cnt + neg_cnt

    pos_prior_prob = (pos_cnt / total_data)
    neg_prior_prob = (neg_cnt / total_data)

    prior_prob = {'pos': pos_prior_prob, 'neg': neg_prior_prob}

    return prior_prob
```

After training the NBC model, we can use this model to predict the label of the testing dataset. The testing dataset needs to be pre-processed first just like the training dataset. The steps of the pre-processing are exactly the same as the training dataset. We use the processed data and the trained NBC model to predict the sentiment of reviews from the words that are inside the reviews. Predicting the label of the testing dataset is done in line 29-55. In order to do this, we calculate the probability of this review to be positive or negative by using the likelihood probability and the prior probability of the model that we have trained. If the positive probability is larger than the negative probability, the review will be predicted as positive sentiment, else, negative sentiment. To calculate it we can use the formula of:

$$P(\theta|D) = P(\theta) \times \prod_{i=1}^{n} p(\mathbf{x}_i|c_i\theta)$$

This formula means that we multiply the prior probability with the product of the likelihood probability of every word with respect to the sentiment, if the word exists in the word features that we have decided. If it does not exist, we can just skip the word. For example, there are “food”, “place”, “like” in the text, then we multiply each likelihood probability of the word with respect to the sentiment, and we multiply it again with the prior probability of the sentiment. In the “predict.py” file, we calculate the probability using the *calc_prob()* function in line 10-17. After getting the predicted label using this way, we calculate the total accuracy of our model. We compare the predicted label with the true label of the testing dataset. There are four possibilities:

1. The true label is positive, and the predicted label is positive as well. This is called true positive (TP).
2. The true label is negative, and the predicted label is negative as well. This is called true negative (TN).
3. The true label is positive but the predicted label is negative. This is called false negative (FN).
4. The true label is negative but the predicted label is positive. This is called false positive (FP).

The formula that we use to calculate the accuracy of the model is as follow:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

The function *calc_accuracy()* (line 6-8) works in this way, and we return the accuracy of the model.

predict.py > predict

```
1   '''
2       This file is to predict the test data,
3       from a trained NBC model
4   '''
5
6   # function to calculate the accuracy
7   def calc_accuracy(acc_count: dict):
8       return ((acc_count['tn'] + acc_count['tp']) / sum(acc_count.values()))
9
10  def calc_prob(words: list, likelihood: dict, prior_prob):
11      prob = prior_prob
12
13      for word in words:
14          if word in likelihood.keys():
15              prob *= likelihood[word]
16
17      return prob
18
19  # function to predict the label of test data
20  def predict(likelihood_pos: dict, likelihood_neg: dict, prior_prob: dict,
21              test_data: list):
22      # to calculate the accuracy using:
23      # tp = true positive
24      # tn = true negative
25      # fp = false positive
26      # fn = false negative
27      acc_count = {'tp': 0, 'tn': 0, 'fp': 0, 'fn': 0}
28
29      # start predicting the label
30      for row in test_data:
31          true_label = int(row['stars'].strip())
32
33          # calculate the positive sentiment probability
34          pos_prob = calc_prob(row['text'], likelihood_pos, prior_prob['pos'])
35
36          # calculate the negative sentiment probability
37          neg_prob = calc_prob(row['text'], likelihood_neg, prior_prob['neg'])
38
39          # predict the label using the probability
40          if pos_prob > neg_prob:
41              pred_label = 5
42          else:
43              pred_label = 1
44
```

predict.py > predict

```
20  def predict(likelihood_pos: dict, likelihood_neg: dict, prior_prob: dict,
45          # add the accuracy
46          if pred_label == 5:
47              if pred_label == true_label:
48                  acc_count['tp'] += 1
49              else:
50                  acc_count['fp'] += 1
51          else:
52              if pred_label == true_label:
53                  acc_count['tn'] += 1
54              else:
55                  acc_count['fn'] += 1
56
57      # calculate the accuracy and return it
58      return calc_accuracy(acc_count)
```

In order to analyze the accuracy of the NBC model with the difference of the amount in training dataset used, we plot the it using *matplotlib* library in Python inside the "plotting.py" file. The *plot()* function (line 7-22) is simply to plot all of the accuracy that we have received from the model and the amount of training dataset utilized to train the model. We also add the annotations in each dot to visualize it easier (line 16-19).

plotting.py > ...

```python
'''
    The file to plot the result of the accuracy
'''

import matplotlib.pyplot as plt

def plot(accuracy: dict):
    plt.figure(figsize=(12, 8))

    # plot the values of the accuracy
    plt.plot(accuracy.keys(), accuracy.values(), marker='o', linestyle='-')

    plt.xlabel('Training Data Set (%)')
    plt.ylabel('Accuracy')

    # set the label at each point
    for i, (xi, yi) in enumerate(zip(accuracy.keys(), accuracy.values())):
        plt.annotate(f'({xi:.0f}%, {yi})', (xi, yi), textcoords="offset points",
                     xytext=(0, 8), ha='center')

    # show the line chart
    plt.show()
```

There is also the “main.py” file to drive all of these codes into one. Also, in the “main.py” file, we printed out the top 20-50 most common words, and save it into “top_20_50_words.txt” file. This file consists all of the top 20-50 most common words from each trial of training the NBC model. The text file will look like the followings:

```
Training Data Set Use: 10%
20. down: 79
21. am: 77
22. minutes: 75
23. restaurant: 75
24. ordered: 74
25. people: 74
26. best: 73
27. day: 72
28. went: 72
29. asked: 72
30. order: 71
31. little: 70
32. still: 70
33. well: 68
34. first: 67
35. take: 64
36. staff: 62
37. going: 61
38. try: 60
39. way: 59
40. came: 58
41. table: 55
42. made: 54
43. car: 53
44. experience: 53
45. years: 53
46. see: 52
47. sure: 51
48. chicken: 51
49. now: 50
50. 2: 50
```

top_20_50_words.txt

```
Training Data Set Use: 30%
20. people: 218
21. didn: 209
22. who: 209
23. love: 206
24. am: 197
25. going: 195
26. down: 193
27. staff: 188
28. ordered: 188
29. well: 187
30. minutes: 182
31. restaurant: 180
32. little: 177
33. day: 176
34. way: 175
35. asked: 175
36. take: 172
37. still: 171
38. first: 168
39. try: 167
40. 2: 166
41. made: 166
42. car: 163
43. came: 161
44. now: 161
45. experience: 161
46. chicken: 161
47. see: 155
48. eat: 151
49. friendly: 146
50. store: 145
```

top_20_50_words.txt

```
Training Data Set Use: 50%
20. best: 360
21. well: 358
22. who: 349
23. going: 331
24. love: 330
25. minutes: 329
26. staff: 327
27. didn: 327
28. am: 310
29. way: 309
30. ordered: 304
31. first: 303
32. restaurant: 301
33. down: 300
34. now: 300
35. chicken: 298
36. day: 292
37. asked: 284
38. came: 281
39. little: 274
40. take: 272
41. 2: 270
42. try: 267
43. experience: 265
44. see: 262
45. made: 260
46. still: 259
47. friendly: 258
48. store: 255
49. eat: 252
50. car: 248
```

top_20_50_words.txt

```
Training Data Set Use: 70%
20. who: 504
21. well: 499
22. best: 495
23. didn: 490
24. going: 485
25. minutes: 457
26. love: 456
27. staff: 454
28. first: 453
29. am: 442
30. down: 440
31. way: 432
32. now: 432
33. day: 418
34. ordered: 416
35. restaurant: 401
36. 2: 399
37. came: 396
38. see: 395
39. take: 394
40. asked: 391
41. car: 389
42. still: 387
43. chicken: 383
44. little: 380
45. nice: 370
46. experience: 363
47. made: 362
48. try: 360
49. friendly: 360
50. store: 358
```

top_20_50_words.txt

```
Training Data Set Use: 100%
20. well: 714
21. order: 703
22. told: 699
23. didn: 691
24. going: 686
25. first: 656
26. am: 651
27. love: 643
28. down: 626
29. staff: 624
30. minutes: 618
31. ordered: 616
32. now: 611
33. way: 599
34. day: 579
35. chicken: 577
36. restaurant: 567
37. came: 558
38. 2: 558
39. nice: 555
40. car: 550
41. take: 544
42. still: 543
43. see: 542
44. asked: 534
45. little: 528
46. store: 517
47. made: 513
48. try: 511
49. want: 507
50. experience: 502
```

Below is also the screenshot of the “main.py” file:

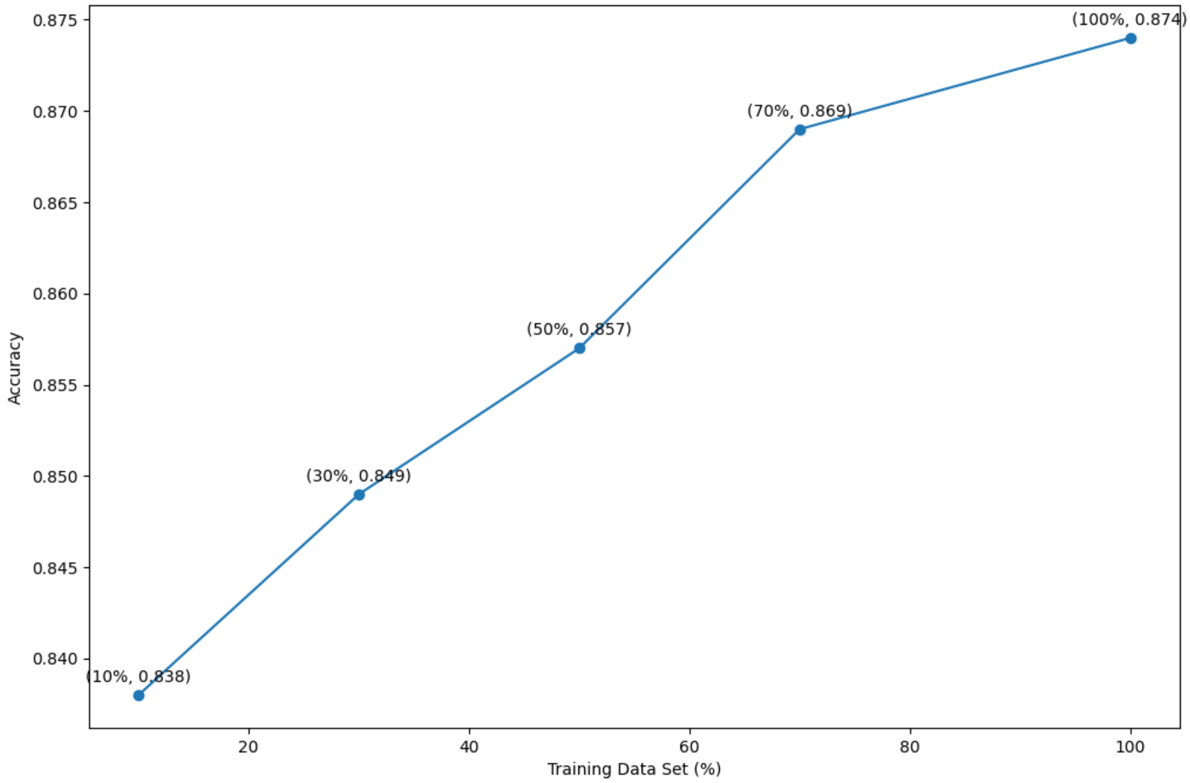
main.py > ...

```python
'''
    Main file to drive all of the code.
    Run this file to train the NBC model
    and predict the test dataset using
    the trained NBC model.
'''

from training import train
from predict import predict
from preprocess import preprocess
from plotting import plot

import os

# specify the data path
train_data_path = "./data/train.csv"
test_data_path = "./data/test.csv"

# the dictionary to store the accuracy of the NBC model
accuracy = {}

# the amount of data to be used to train the NBC model
use_train_data_percentage = [0.1, 0.3, 0.5, 0.7, 1]

# pre-process the test data first
test_data = preprocess(test_data_path)[0]

top_20_50_file = "top_20_50_words.txt"

# remove the text file first if exists
if os.path.exists(top_20_50_file):
    os.remove(top_20_50_file)

# the loop to train the NBC and predict the test data
# and calculate the accuracy of the model
for i in use_train_data_percentage:
    # pre-process the train data first
    train_data, train_words_label_count, train_words_list = preprocess(train_data_path, i)

    # get the top 50 words that appear
    top_twenty_fifty = dict(sorted(train_words_list.items(), key=lambda item: item[1], reverse=True)[19:50])
```

main.py > ...

```python
    count = 20

    # save the top 50 words in a file
    fh = open(top_20_50_file, 'a')
    fh.write(f"Training Data Set Use: {(i * 100):.0f}%\n")
    for k, v in top_twenty_fifty.items():
        fh.write(f'{count}. {k}: {v}\n')
        count += 1

    fh.write("\n\n")
    fh.flush()
    fh.close()

    # train the NBC model
    likelihood_pos, likelihood_neg, prior_prob = train(train_data, train_words_label_count, train_words_list)

    # predict the label from the trained NBC and calculate the accuracy of the model
    temp_accuracy = predict(likelihood_pos, likelihood_neg, prior_prob, test_data)

    # save the accuracy
    accuracy[(i * 100)] = temp_accuracy

plot(accuracy)
```

Using the codes above that we have created, the learning curve that we were able to get was:



As we can see in the line chart above, the more training dataset that we use the better our NBC model is, shown by the increasing trend of the accuracy. This happens because, our model will be able to learn more if there are more data available for it to use as its learning platform. By having more data, the model could learn the pattern more, and thus, provide a better prediction. The lesser the data is available, the poorer the model will be. We believe that this is not happening in terms of NBC model only, but almost to every machine learning model. The more data that it can learn from, the better model it will produce. Specifically in this project, more word features could be produced from having more data, giving the model to learn and understand more of the positive and negative sentiment pattern based on the text reviews.

In conclusion, to have a good NBC model, it required a lot of data, because the more training data it can have, the better model it will produce. Hence, it will have a more accurate prediction towards unknown data.