Metehan Kaya
21401258
Section 2

HOMEWORK 1

1. The Chess Game

Question 1.1

Since Gates plays *bold*, he will either *win* or *lose* each game. Therefore, at the end of first 3 games, one of the players will win strictly more games than the other player. A player wins the match if he wins at least 2 games.

$$P(G\ wins\ the\ match) = P(G\ wins\ 2\ games\ out\ of\ 3) + P(G\ wins\ 3\ games\ out\ of\ 3)$$

$$= \binom{3}{2} * P(G\ wins\ a\ game)^2 * P(G\ loses\ a\ game)^1 + \binom{3}{3} * P(G\ wins\ a\ game)^3$$

$$= 3 * 0.6^2 * 0.4^1 + 1 * 0.6^3 = 0.648$$

Note: G stands for Gates, S stands for Steve.

Question 1.2

Since Gates plays *timid*, he will either lose or draw each game. Therefore, during the first 3 games, he should NOT lose any game and return *bold* to win the 4th game, and the match.

$$P(G\ wins\ the\ match) = P(G\ draws\ first\ 3\ games) * P\left(G\ wins\ the\ 4^{th}\ game\right)$$

$$= P(G\ draws\ a\ game \mid timid)^3 * P(G\ wins\ a\ game \mid bold) = 0.65^3 * 0.6 = 0.164775$$

Question 1.3

Since G chooses styles randomly, $P(style = bold) = P(style = timid) = 1/2$.

$$P(style = bold \mid game = lose) = \frac{P(style = bold\ \&\ game = lose)}{P(game = lose)}$$

$$= \frac{P(style = bold\ \&\ game = lose)}{P(game = lose \mid style = bold) * P(style = bold) + P(game = lose \mid style = timid) * P(style = timid)}$$

$$= \frac{\frac{1}{2} * 0.4}{\frac{1}{2} * 0.4 + \frac{1}{2} * 0.35} = \frac{8}{15} = 0.5\overline{3}$$

Question 1.4

Steve may guess Gates's strategy correctly or not. Therefore, both cases should be considered.

$$P(S\ wins\ a\ game) = P(S = win\ \&\ prediction = +) + P(S = win\ \&\ prediction = -)$$

$$= P(S = win \mid p = +) * P(p = +) + \left(1 - P(p = +)\right) * (P(style = timid\ \&\ S = win) + P(style = bold\ \&\ S = win))$$

$$= 0.85 * 0.75 + 0.25 * (\frac{1}{2} * 0.35 + \frac{1}{2} * 0.4) = 0.73125$$

Metehan Kaya
21401258
Section 2

2. Medical Diagnosis

Question 2.1

$$P(S = disease) = \frac{5}{1000} = \frac{1}{200}\ (already\ given)$$

$$P(S = healthy) = 1 - P(S = disease) = 1 - \frac{1}{200} = \frac{199}{200}$$

$$P(T = + \mid S = disease) = \frac{97}{100}\ (already\ given)$$

$$P(T = - \mid S = disease) = 1 - P(T = + \mid S = disease) = 1 - \frac{97}{100} = \frac{3}{100}$$

$$P(T = + \mid S = healthy) = 1 - P(T = - \mid S = healthy) = 1 - \frac{98}{100} = \frac{2}{100}$$

$$P(T = - \mid S = healthy) = \frac{98}{100}\ (already\ given)$$

Question 2.2

$$P(S = disease \mid T = +) = \frac{P(S = disease\ \&\ T = +)}{P(T = +)}$$

$$= \frac{P(S = disease) * P(T = + \mid S = disease)}{P(S = disease) * P(T = + \mid S = disease) +\ P(S = healthy) * P(T = + \mid S = healthy)}$$

$$= \left(\frac{1}{200} * \frac{97}{100}\right) \Big/ \left(\frac{1}{200} * \frac{97}{100} + \frac{199}{200} * \frac{2}{100}\right) = \frac{97}{97 + 398} = \frac{97}{495} = 0.1\overline{95} \approx 0.2$$

Given that the prediction is positive, the probability of the disease is close to 20% is smaller than 50%. Therefore, patient should be considered as healthy.

3. MLE and MAP

Question 3.1

Given λ, the probability of $X = \{x_1, x_2, x_3, \dots, x_n\}$ is given below.

$$P(X_1 = x_1, \dots, X_n = x_n \mid \lambda) = \prod_{i=1}^{n} \frac{\lambda^{x_i} * e^{-\lambda}}{x_i!}$$

Taking logarithm of it, we get the summation below.

$$\sum_{i=1}^{n} -\lambda + x_i * \log(\lambda) - \log(x_i!)$$

Taking derivative of it with respect to λ, we get the MLE of λ.

$$-n + \sum_{i=1}^{n} \frac{x_i}{\lambda} = 0$$

Metehan Kaya
21401258
Section 2

$$\hat{\lambda} = \frac{\sum_{i=1}^{n} x_i}{n}$$

This means that $\hat{\lambda} = \arg max\, P(X|\lambda) = Mean\ of\ x_i s.$

Question 3.2

$$\hat{\lambda} = \arg max \frac{P(X \mid \lambda) * P(\lambda)}{P(X)}$$

Taking logarithm of it, we get the expression below.

$$(\alpha * \log(\beta) + (\alpha - 1) * \log(\lambda) - \beta * \lambda - \log \Gamma(\alpha)) - \log(P(X)) \sum_{i=1}^{n} -\lambda + x_i * \log(\lambda) - \log(x_i!)$$

Taking derivative of it with respect to λ, we get the MAP of λ.

$$\frac{\alpha - 1}{\lambda} - \beta - n + \frac{\sum_{i=1}^{n} x_i}{\lambda} = 0$$

$$\hat{\lambda} = \frac{(\alpha - 1) + \sum_{i=1}^{n} x_i}{\beta + n}$$

Question 3.3

$$\hat{\lambda}_{MAP} = \arg max \frac{P(X \mid \lambda) * P(\lambda)}{P(X)} = \arg max\, P(X \mid \lambda) * P(\lambda) = \arg max\, P(X \mid \lambda) = \hat{\lambda}_{MLE}$$

The reason behind the first reduction is that X is given. The reason behind the second reduction is that $P(\lambda)$ is constant since $\lambda \sim U(a, b)$. Therefore, MLE estimate of λ and MAP estimate of λ with the uniform prior is the same for any (a, b).

4. Sentiment Analysis on Tweets

Question 4.1

For each tweet, we aim to find a class that maximizes the given formula. For each class, the denominator stays the same. In other words, the denominator has the same effect on each candidate class. Hence, we can ignore it and just look for the numerator.

Question 4.2

In the training dataset, $N_{neutral} = 2617$, $N_{positive} = 2004$ and $N_{negative} = 7091$ out of $N = 11712$ tweets. $\pi_{neutral} \approx 0.223$, $\pi_{positive} \approx 0.171$ and $\pi_{negative} \approx 0.605$. As it is seen, the training set is skewed towards the negative class.

Considering Eq. (4.6), density of negative class does NOT affect summation part since denominator of theta values deals with this. Also, $t_{wj,i}$ is related to length of a tweet, not a class. However, left part of the formula changes according to the π value which denotes density of a class. Since the negative class dominates the training data, and therefore the formula, it will be harder to detect positive and negative classes. Also, suppose that another class dominates the test data. In that case, *false positive* (*FP*) for negative class would be high.

Metehan Kaya
21401258
Section 2

Many solutions can be proposed to this problem. However, guessing which one is the best is a little bit difficult. Effect of π can be removed, a new algorithm and formula can be generated, new training data can be added, training data can be separated to different classes and same number of tweets can be chosen for each class

Question 4.3

Simply, we need $n_{class} = 3$ for π and $n_{class} * n_{word} = 3 * 5722 = 17166$ for θ. However, if I assume that I know $n_{tweet} = 11712$, then I can predict P(negative) since π_i values sum up to 1.

$$P(negative) = 1 - (P(positive) + P(neutral))$$

Therefore, $17166 + 3 - 1 = 17168$ parameters are needed.

Question 4.4

Out of 2928 tests, $\#\ correct\ prediction = 1839$ and $\#\ wrong\ prediction = 1089$.

Also, $accuracy \approx 0.628$. For each pair of predicted and actual class, number of tests can be seen below.

Prediction / Actual	A = neutral	A = positive	A = negative
P = neutral	207	107	500
P = positive	18	80	35
P = negative	257	172	1552

Table 1. Multinomial Naïve Bayes classifier with MLE estimation

Considering Eq. (4.6), in the case of α*log(0), value in the equation becomes *-inf*. If a tweet encounters this issue for all classes, then it will be harder to compare and take maximum among them. As a solution, I made slight changes in the method and counted *-inf* for each candidate class to predict the class of a given specific tweet. I prioritized the number of *-inf* and take the minimum among candidate classes. Then, I observed $\#\ correct\ prediction' = 2138$ and $accuracy' \approx 0.73$.

It seems that MLE estimation has a terrible accuracy. If the label *negative* was chosen, we would have $\#\ correct\ prediction'' = 2087$ and $accuracy'' \approx 0.713 > 0.628$. However, this is only one of the metrics used to understand how good the estimation is. For instance, for positive and neutral tweets, precision and recall would be zero since there will be no *true positive* (*TP*) for positive and neutral tweets.

Question 4.5

Out of 2928 tests, $\#\ correct\ prediction = 2205$ and $\#\ wrong\ prediction = 723$.

Also, $accuracy \approx 0.753$. For each pair of predicted and actual class, number of tests can be seen below.

Prediction / Actual	A = neutral	A = positive	A = negative
P = neutral	169	46	131
P = positive	20	123	43
P = negative	293	190	1913

Table 2. Multinomial Naïve Bayes classifier with MAP estimation

Since a value $\alpha > 1$ is added to the maximization problem, there will be no issue originating from *-inf*. It will be impossible for a θ value to be 0. Therefore, the predictions will be better. As a sign of it, increase in accuracy by 13% can be given. Compared to Multinomial Naïve Bayes classifier with MLE estimation, *true positive* (*TP*) of positive and negative class increases where it decreases for neutral class. In my opinion, the reason behind this is that in case of ties, the predicted class was neutral for MLE estimation and we don't face with this condition when using MAP estimation. Therefore, this leads to less amount of correctly predicted neutral tweets.

Question 4.6

Out of 2928 tests, $\#\ correct\ prediction = 1878$ and $\#\ wrong\ prediction = 1050$.

Also, $accuracy \approx 0.641$. For each pair of predicted and actual class, number of tests can be seen below.

Prediction / Actual	A = neutral	A = positive	A = negative
P = neutral	267	160	515
P = positive	25	85	46
P = negative	190	114	1526

Table 3. Bernoulli Naïve Bayes classifier with MLE estimation

A similar situation to the Q4.4 can be seen here: $accuracy' \approx 0.713 > 0.641$. Also, it can be claimed that Bernoulli Naïve Bayes is better than Multinomial Naïve Bayes used before. It seems that whatever approach we use to learn, as long as MLE estimation is used, there is no a massive change in the accuracy. I made the similar change what I did at Multinomial Naïve Bayes with MLE estimation, counted number of *-inf*. Then, I observed $\#\ correct\ prediction'' = 2175$ and $accuracy'' \approx 0.743$.

Compared to Multinomial Naïve Bayes with MLE estimation, the number of (correctly) predicted neutral and (correctly) predicted positive tweets increase whereas (correctly) predicted negative tweets decrease. The accuracy increases by 1.2% which means instead of counting the same word multiple times, checking the existence of it is more useful.

Question 4.7

For each class, words exist in the vocabulary list are listed and sorted by frequency in non-increasing order.

Neutral: ['@jetblue', '@united', '@southwestair', 'flight', '@usairways', '@virginamerica', 'flights', 'help', 'fleek', "fleet's", 'dm', 'tomorrow', 'time', 'flying', 'cancelled', 'fly', 'change', 'travel', 'today', 'check']

Positive: ['@southwestair', '@jetblue', '@united', 'flight', '@usairways', 'great', '@virginamerica', 'service', 'love', 'best', 'guys', 'customer', 'time', 'awesome', 'airline', 'help', 'amazing', 'today', 'fly', 'flying']

Negative: ['@united', 'flight', '@usairways', '@southwestair', '@jetblue', 'cancelled', 'service', 'hours', 'hold', 'time', 'customer', 'help', 'delayed', 'plane', 'hour', 'flights', 'bag', 'gate', 'late', 'flightled']

Given lists have many words in common; including twitter usernames with @s, some simple words related to the airlines in U.S like *flight*, and words with suffixes like *flying*. From my perspective, the words that don't exist in the actual English vocabulary, common technical words related to the domain and different words with the same origin make harder to understand what label is related with what words.

Among the most commonly used 20 words in positive tweets, only 5 of them (*great*, *love*, *best*, *awesome* and *amazing*) are related to positivity. Among the most commonly used 20 words in negative tweets, only 3 of them (*cancelled*, *delayed* and *late*) are related to negativity. Having a few keywords is a sign of how bad this approach and vocabulary are. Among the most commonly used 20 words in neutral tweets, 2 of them (*fleek* and *cancelled*) are related to other classes and it seems good actually.

As a solution, the words existing in the English vocabulary can be considered, and among them, the ones with low IDF (Inverse Document Frequency) can be removed. In that way, the words occurring too much in any case will NOT occupy a good candidate word. However, this may affect prediction of neutral tweets in a bad way because the words like *flight* would be eliminated. Therefore, it would be better not to use this method for prediction of neutral tweets.

5. Code

5.1 main.py (Main source code to run)

```
from read_data import get_inputs

from helper import get_label_counter
from helper import get_train_pi
from helper import get_train_t
from helper import get_train_theta_t
from helper import get_train_theta_t_map
from helper import get_train_s
from helper import get_train_theta_s

from tester import multinomial_naive_bayes
from tester import bernoulli_naive_bayes
from tester import find_accuracy

from observe_common import get_frequent_word_ids
from observe_common import read_vocab_file


train_labels, train_features_matrix, test_labels, test_features_matrix =
get_inputs()

train_n_neutral, train_n_positive, train_n_negative =
get_label_counter(train_labels)
print('train_n')
print(train_n_neutral, train_n_positive, train_n_negative)

train_pi_neutral, train_pi_positive, train_pi_negative =
get_train_pi(train_n_neutral, train_n_positive, train_n_negative)
train_pis = [train_pi_neutral, train_pi_positive, train_pi_negative]
print('train_pis')
print(train_pis)

train_t, train_t_total = get_train_t(train_labels, train_features_matrix)
print('train_t')
print(train_t)
print('train_t_total')
print(train_t_total)

train_theta_t = get_train_theta_t(train_t, train_t_total)
print('train_theta_t')
print(train_theta_t)
```

```
train_theta_t_map = get_train_theta_t_map(train_t, train_t_total)
# print('train_theta_t_map')
# print(train_theta_t_map)

train_s = get_train_s(train_labels, train_features_matrix)
print('train_s')
print(train_s)

train_theta_s = get_train_theta_s(train_s, train_n_neutral,
train_n_positive, train_n_negative)
print('train_theta_s')
print(train_theta_s)

test_naive_bayes = multinomial_naive_bayes(train_pis, test_features_matrix,
train_theta_t)
print('test_naive_bayes')
print(test_naive_bayes)
find_accuracy(test_naive_bayes, test_labels)

test_naive_bayes_map = multinomial_naive_bayes(train_pis,
test_features_matrix, train_theta_t_map)
print('test_naive_bayes_map')
print(test_naive_bayes_map)
find_accuracy(test_naive_bayes_map, test_labels)

test_bernoulli_naive_bayes = bernoulli_naive_bayes(train_pis,
test_features_matrix, train_theta_s)
print('test_bernoulli_naive_bayes')
print(test_bernoulli_naive_bayes)
find_accuracy(test_bernoulli_naive_bayes, test_labels)

vocab = read_vocab_file()
print('vocab')
print(vocab)

all_freq_words = get_frequent_word_ids(train_t, vocab)
print('all_freq_word_ids')
print('Neutral:', all_freq_words[0])
print('Positive:', all_freq_words[1])
print('Negative:', all_freq_words[2])
```

5.2 helper.py (Deals with calculation of parameters related to training)

```
import os
import pickle


def get_label_counter(train_labels):
    pickle_file_name = 'train_n_label.pkl'
    if not os.path.isfile(pickle_file_name):
        train_n_neutral = 0
        train_n_positive = 0
        train_n_negative = 0
        for i in range(len(train_labels)):
            if train_labels[i] == 'neutral':
                train_n_neutral += 1
            elif train_labels[i] == 'positive':
                train_n_positive += 1
            elif train_labels[i] == 'negative':
                train_n_negative += 1
        pickle.dump([train_n_neutral, train_n_positive, train_n_negative],
```

```
open(pickle_file_name, "wb"))
    else:
        with open(pickle_file_name, 'rb') as pickle_file:
            train_n_neutral, train_n_positive, train_n_negative = 
pickle.load(pickle_file)
    return [train_n_neutral, train_n_positive, train_n_negative]


def get_train_pi(train_n_neutral, train_n_positive, train_n_negative):
    pickle_file_name = 'train_pi.pkl'
    if not os.path.isfile(pickle_file_name):
        sum = train_n_neutral + train_n_positive + train_n_negative
        train_pi_neutral = float(train_n_neutral) / sum
        train_pi_positive = float(train_n_positive) / sum
        train_pi_negative = float(train_n_negative) / sum
        pickle.dump([train_pi_neutral, train_pi_positive, 
train_pi_negative], open(pickle_file_name, "wb"))
    else:
        with open(pickle_file_name, 'rb') as pickle_file:
            train_pi_neutral, train_pi_positive, train_pi_negative = 
pickle.load(pickle_file)
    return [train_pi_neutral, train_pi_positive, train_pi_negative]


def get_train_t(train_labels, train_features_matrix):
    pickle_file_name = 'train_t.pkl'
    if not os.path.isfile(pickle_file_name):
        n_tweet = len(train_labels)
        n_feature = len(train_features_matrix[0])
        train_t_total = [0, 0, 0]
        train_t = []
        for i in range(n_feature):
            train_t.append([0, 0, 0])
        for tweet in range(n_tweet):
            if train_labels[tweet] == 'neutral':
                label_id = 0
            elif train_labels[tweet] == 'positive':
                label_id = 1
            elif train_labels[tweet] == 'negative':
                label_id = 2
            for feature in range(n_feature):
                feature_cnt = train_features_matrix[tweet][feature]
                train_t[feature][label_id] += feature_cnt
                train_t_total[label_id] += feature_cnt
        pickle.dump([train_t, train_t_total], open(pickle_file_name, "wb"))
    else:
        with open(pickle_file_name, 'rb') as pickle_file:
            train_t, train_t_total = pickle.load(pickle_file)
    return [train_t, train_t_total]


def get_train_theta_t(train_t, train_t_total):
    pickle_file_name = 'train_theta_t.pkl'
    if not os.path.isfile(pickle_file_name):
        train_theta_t = []
        n_feature = len(train_t)
        for feature in range(n_feature):
            train_theta_t_neutral = float(train_t[feature][0]) / 
train_t_total[0]
            train_theta_t_positive = float(train_t[feature][1]) /
```

```
train_t_total[1]
            train_theta_t_negative = float(train_t[feature][2]) /
train_t_total[2]
            train_theta_t.append([train_theta_t_neutral,
train_theta_t_positive, train_theta_t_negative])
        pickle.dump(train_theta_t, open(pickle_file_name, "wb"))
    else:
        with open(pickle_file_name, 'rb') as pickle_file:
            train_theta_t = pickle.load(pickle_file)
    return train_theta_t


def get_train_theta_t_map(train_t, train_t_total, alpha=1):
    pickle_file_name = 'train_theta_t_map.pkl'
    if not os.path.isfile(pickle_file_name):
        train_theta_t_map = []
        n_feature = len(train_t)
        for feature in range(n_feature):
            train_theta_t_map_neutral = float(train_t[feature][0] + alpha)
/ (train_t_total[0] + alpha * n_feature)
            train_theta_t_map_positive = float(train_t[feature][1] + alpha)
/ (train_t_total[1] + alpha * n_feature)
            train_theta_t_map_negative = float(train_t[feature][2] + alpha)
/ (train_t_total[2] + alpha * n_feature)
            train_theta_t_map.append([train_theta_t_map_neutral,
train_theta_t_map_positive, train_theta_t_map_negative])
        pickle.dump(train_theta_t_map, open(pickle_file_name, "wb"))
    else:
        with open(pickle_file_name, 'rb') as pickle_file:
            train_theta_t_map = pickle.load(pickle_file)
    return train_theta_t_map


def get_train_s(train_labels, train_features_matrix):
    pickle_file_name = 'train_s.pkl'
    if not os.path.isfile(pickle_file_name):
        n_tweet = len(train_labels)
        n_feature = len(train_features_matrix[0])
        train_s = []
        for i in range(n_feature):
            train_s.append([0, 0, 0])
        for tweet in range(n_tweet):
            if train_labels[tweet] == 'neutral':
                label_id = 0
            elif train_labels[tweet] == 'positive':
                label_id = 1
            elif train_labels[tweet] == 'negative':
                label_id = 2
            for feature in range(n_feature):
                feature_cnt = min(train_features_matrix[tweet][feature], 1)
                train_s[feature][label_id] += feature_cnt
        pickle.dump(train_s, open(pickle_file_name, "wb"))
    else:
        with open(pickle_file_name, 'rb') as pickle_file:
            train_s = pickle.load(pickle_file)
    return train_s


def get_train_theta_s(train_s, train_n_neutral, train_n_positive,
train_n_negative):
```

```
    pickle_file_name = 'train_theta_s.pkl'
    if not os.path.isfile(pickle_file_name):
        train_theta_s = []
        n_feature = len(train_s)
        for feature in range(n_feature):
            train_theta_s_neutral = float(train_s[feature][0]) /
train_n_neutral
            train_theta_s_positive = float(train_s[feature][1]) /
train_n_positive
            train_theta_s_negative = float(train_s[feature][2]) /
train_n_negative
            train_theta_s.append([train_theta_s_neutral,
train_theta_s_positive, train_theta_s_negative])
        pickle.dump(train_theta_s, open(pickle_file_name, "wb"))
    else:
        with open(pickle_file_name, 'rb') as pickle_file:
            train_theta_s = pickle.load(pickle_file)
    return train_theta_s
```

5.3 tester.py (Tests the model, calculates accuracy, the functions counting -*inf* are commented out)

```
import math


def multinomial_naive_bayes(train_pis, test_features_matrix,
train_theta_t):
    n_tweet = len(test_features_matrix)
    n_feature = len(test_features_matrix[0])
    test_result = []
    for tweet in range(n_tweet):
        score = [0, 0, 0]
        for k in range(3):
            score[k] = math.log(train_pis[k])
            for feature in range(n_feature):
                theta = train_theta_t[feature][k]
                test_feature = test_features_matrix[tweet][feature]
                if theta == 0:
                    if test_feature == 0:
                        score[k] += 0
                    else:
                        score[k] += -math.inf
                else:
                    score[k] += test_feature * math.log(theta)
        best_label = -1
        best_score = -1
        for k in range(3):
            if score[k] > best_score or best_label == -1:
                best_score = score[k]
                best_label = k
        if math.fabs(score[1] - score[2]) < 0.000001:
            best_label = 0
        if best_label == 0:
            test_result.append('neutral')
        elif best_label == 1:
            test_result.append('positive')
        elif best_label == 2:
            test_result.append('negative')
    return test_result


'''
```

```
def naive_bayes(train_pis, test_features_matrix, train_theta_t):
    n_tweet = len(test_features_matrix)
    n_feature = len(test_features_matrix[0])
    test_result = []
    for tweet in range(n_tweet):
        score = [0, 0, 0]
        minf = [0, 0, 0]
        for k in range(3):
            score[k] = math.log(train_pis[k])
            for feature in range(n_feature):
                theta = train_theta_t[feature][k]
                test_feature = test_features_matrix[tweet][feature]
                if theta == 0:
                    if test_feature == 0:
                        score[k] += 0
                    else:
                        minf[k] += 1
                else:
                    score[k] += test_feature * math.log(theta)
        best_label = -1
        best_score = -1
        best_minf = n_feature + 5
        for k in range(3):
            if best_label == -1 or (minf[k] < best_minf) or (minf[k] ==
best_minf and score[k] > best_score):
                best_minf = minf[k]
                best_score = score[k]
                best_label = k
        if minf[1] == minf[2] and math.fabs(score[1] - score[2]) <
0.000001:
            best_label = 0
        if best_label == 0:
            test_result.append('neutral')
        elif best_label == 1:
            test_result.append('positive')
        elif best_label == 2:
            test_result.append('negative')
    return test_result
'''


def bernoulli_naive_bayes(train_pis, test_features_matrix, train_theta_s):
    n_tweet = len(test_features_matrix)
    n_feature = len(test_features_matrix[0])
    test_result = []
    for tweet in range(n_tweet):
        score = [0, 0, 0]
        for k in range(3):
            score[k] = math.log(train_pis[k])
            for feature in range(n_feature):
                theta = train_theta_s[feature][k]
                test_feature = min(test_features_matrix[tweet][feature], 1)
                value = test_feature * theta + (1-test_feature) * (1-theta)
                if value == 0:
                    score[k] += -math.inf
                else:
                    score[k] += math.log(value)
        best_label = -1
        best_score = -1
        for k in range(3):
```

```
            if best_label == -1 or score[k] > best_score:
                best_score = score[k]
                best_label = k
        if math.fabs(score[1] - score[2]) < 0.000001:
            best_label = 0
        if best_label == 0:
            test_result.append('neutral')
        elif best_label == 1:
            test_result.append('positive')
        elif best_label == 2:
            test_result.append('negative')
    return test_result


'''
def bernoulli_naive_bayes(train_pis, test_features_matrix, train_theta_s):
    n_tweet = len(test_features_matrix)
    n_feature = len(test_features_matrix[0])
    test_result = []
    for tweet in range(n_tweet):
        score = [0, 0, 0]
        score_log = [0, 0, 0]
        minf = [0, 0, 0]
        for k in range(3):
            score[k] = math.log(train_pis[k])
            for feature in range(n_feature):
                theta = train_theta_s[feature][k]
                test_feature = min(test_features_matrix[tweet][feature], 1)
                value = test_feature * theta + (1-test_feature) * (1-theta)
                if value == 0:
                    minf[k] += 1
                else:
                    score_log[k] += math.log(value)
            score[k] += score_log[k]
        best_label = -1
        best_score = -1
        best_minf = n_feature + 5
        for k in range(3):
            if best_label == -1 or (minf[k] < best_minf) or (minf[k] ==
best_minf and score[k] > best_score):
                best_minf = minf[k]
                best_score = score[k]
                best_label = k
        if minf[1] == minf[2] and math.fabs(score[1] - score[2]) <
0.000001:
            best_label = 0
        if best_label == 0:
            test_result.append('neutral')
        elif best_label == 1:
            test_result.append('positive')
        elif best_label == 2:
            test_result.append('negative')
    return test_result
'''


def get_class_id(label):
    if label == 'neutral':
        return 0
    elif label == 'positive':
```

```
        return 1
    elif label == 'negative':
        return 2
    return -1


def find_accuracy(test_result, test_labels):
    correct = 0
    failure = 0
    counter = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
    n_tweet = len(test_result)
    for tweet in range(n_tweet):
        prediction = get_class_id(test_result[tweet])
        actual = get_class_id(test_labels[tweet])
        counter[prediction][actual] += 1
        if test_result[tweet] == test_labels[tweet]:
            correct += 1
        else:
            failure += 1
    print('correct: ', correct)
    print('failure: ', failure)
    print('accuracy: ', float(correct) / (correct + failure))
    print('counter: ', counter)
```

5.4 read_data.py (Reads 4 csv files and gets the input data)

```
import os
import csv
import pickle


def get_input_file(input_file_name, pickle_file_name, n_cols):
    if not os.path.isfile(pickle_file_name):
        mat = []
        with open(input_file_name) as csv_file:
            csv_reader = csv.reader(csv_file, delimiter=',')
            for row in csv_reader:
                row_data = []
                for col in range(n_cols):
                    if n_cols > 1:
                        row_data.append(int(row[col]))
                    else:
                        mat.append(row[col])
                if n_cols > 1:
                    mat.append(row_data)
        pickle.dump(mat, open(pickle_file_name, "wb"))
    else:
        with open(pickle_file_name, 'rb') as pickle_file:
            mat = pickle.load(pickle_file)
    return mat


def get_inputs():
    train_labels = get_input_file('question-4-train-labels.csv',
'train_labels.pkl', 1)
    train_features_matrix = get_input_file('question-4-train-features.csv',
'train_features_matrix.pkl', 5722)
    test_labels = get_input_file('question-4-test-labels.csv',
'test_labels.pkl', 1)
    test_features_matrix = get_input_file('question-4-test-features.csv',
'test_features_matrix.pkl', 5722)
```

```
    return [train_labels, train_features_matrix, test_labels,
test_features_matrix]
```

5.5 observe_common.py (Separated to Question 4.7)

```
import os
import pickle


def get_frequent_word_ids(train_t, vocab, alpha=20):
    n_word = len(train_t)
    all_freq_words = []
    for k in range(3):
        word_pairs = []
        for word in range(n_word):
            word_pairs.append([word, train_t[word][k]])
        sorted_word_pairs = sorted(word_pairs, key=lambda x: x[1])
        i = n_word - 1
        freq_words = []
        while i >= n_word - alpha:
            freq_words.append(vocab[sorted_word_pairs[i][0]])
            i -= 1
        all_freq_words.append(freq_words)
    return all_freq_words


def read_vocab_file():
    if not os.path.isfile("vocabulary.pkl"):
        vocab = []
        with open("question-4-vocab.txt", encoding="utf8") as ins:
            for line in ins:
                length = len(line)
                index = length-2
                while line[index:index+1].isdigit():
                    index -= 1
                vocab.append(line[0:index])
        pickle.dump(vocab, open("vocabulary.pkl", "wb"))
    else:
        with open('vocabulary.pkl', 'rb') as pickle_file:
            vocab = pickle.load(pickle_file)
    return vocab
```