Homework 6

Due November 5 at 11 pm

1. (Spam detector) In order to build a spam classifier, we gather the following data. Each row is an email. The first column indicates whether it is spam or not. The remaining columns indicate whether it contains (✓) or not (✗) the word on top.

	Miracle	Alternative	Medicine	Basketball
Spam	✓	✓	✓	✓
Not spam	✗	✗	✓	✓
Spam	✓	✗	✗	✗
Not spam	✗	✓	✗	✗
Spam	✓	✗	✗	✗
Not spam	✗	✓	✗	✓
Spam	✓	✗	✗	✗
Spam	✗	✓	✓	✗
Not spam	✓	✓	✗	✓
Not spam	✗	✗	✓	✓

We use a Bernoulli random variable $\tilde{y}$ to model whether the email is spam ($\tilde{y} = 1$) or not ($\tilde{y} = 0$), and a four-dimensional random vector $\tilde{x}$ to indicate whether the ith word is ($\tilde{x}[i] = 1$) or not ($\tilde{x}[i] = 0$) in the email for $i \in \{1, 2, 3, 4\}$.

(a) Your friend recommends that you estimate the conditional pmf of $\tilde{y}$ given $\tilde{x}$ and then maximize it to produce your estimate. What is the problem with this approach?

The problem with this approach is that if we are to compute conditional pmf of $\tilde{y}$ given $\tilde{x}$, we will have to know the joint distribution of $\tilde{x}[1], \tilde{x}[2], \tilde{x}[3], \tilde{x}[4]$. If we are using the empirical pmf to estimate the distribution,we will have $2^4 = 16$ parameters to estimate, which seems ok in this problem but in reality the number of suspicious words in spam may be way lot more, where the parameter to be estimated will explode. Thus the curse of dimensionality will occur.

(b) Apply naive Bayes to classify an email that reads *I hurt my foot playing basketball. Can you get me some medicine?*

We want to find the following:

$$
\begin{aligned}
MAP(\vec{x}) &= \arg\max_{y \in \{Spam, not\ spam\}} p_{\tilde{y}|\tilde{x}}(y|x) \\
&= \arg\max_{y \in \{Spam, not\ spam\}} \frac{\prod_{i=1}^{4} p_{\tilde{x}[i]|\tilde{y}}(x_{[i]}|y) p_{\tilde{y}}(y)}{\sum_{y \in \{Spam, not\ spam\}} \prod_{i=1}^{4} p_{\tilde{x}[i]|\tilde{y}}(x_{[i]}|y) p_{\tilde{y}}(y)} \\
&= \arg\max_{y \in \{Spam, not\ spam\}} \frac{\prod_{i=1}^{2} p_{\tilde{x}[i]|\tilde{y}}(0|y) p_{\tilde{y}}(y) \prod_{i=3}^{4} p_{\tilde{x}[i]|\tilde{y}}(1|y) p_{\tilde{y}}(y)}{\sum_{y \in \{Spam, not\ spam\}} \prod_{i=1}^{2} p_{\tilde{x}[i]|\tilde{y}}(0|y) p_{\tilde{y}}(y) \prod_{i=3}^{4} p_{\tilde{x}[i]|\tilde{y}}(1|y) p_{\tilde{y}}(y)} \\
&= \arg\max_{y \in \{Spam, not\ spam\}} \{ \frac{\frac{1}{2} \times \frac{1}{5} \times \frac{3}{5} \times \frac{2}{5} \times \frac{4}{5}}{\frac{1}{2} \times \frac{1}{5} \times \frac{3}{5} \times \frac{2}{5} \times \frac{4}{5} + \frac{1}{2} \times \frac{4}{5} \times \frac{2}{5} \times \frac{2}{5} \times \frac{4}{5}} \\
&, \frac{\frac{1}{2} \times \frac{4}{5} \times \frac{2}{5} \times \frac{2}{5} \times \frac{4}{5}}{\frac{1}{2} \times \frac{1}{5} \times \frac{3}{5} \times \frac{2}{5} \times \frac{4}{5} + \frac{1}{2} \times \frac{4}{5} \times \frac{2}{5} \times \frac{2}{5} \times \frac{4}{5}} \} \\
&= not\ spam
\end{aligned}
$$

(c) Apply naive Bayes to classify an email that reads *This alternative medicine is amazing, send us all your money!* Explain what shortcoming of the naive Bayes classifier is illustrated by this example.

We want to find the following:

$$
\begin{aligned}
MAP(\vec{x}) &= \underset{y \in \{Spam, not\ \ spam\}}{\arg\max}\ p_{\tilde{y}|\tilde{x}}(y|x) \\
&= \underset{y \in \{Spam, not\ \ spam\}}{\arg\max} \frac{\prod_{i=1}^{4} p_{\tilde{x}[i]|\tilde{y}}(x_{[i]}|y)p_{\tilde{y}}(y)}{\sum_{y \in \{Spam, not\ \ spam\}} \prod_{i=1}^{4} p_{\tilde{x}[i]|\tilde{y}}(x_{[i]}|y)p_{\tilde{y}}(y)} \\
&= \underset{y \in \{Spam, not\ \ spam\}}{\arg\max} \frac{\prod_{i=1,4} p_{\tilde{x}[i]|\tilde{y}}(0|y)p_{\tilde{y}}(y) \prod_{i=2,3} p_{\tilde{x}[i]|\tilde{y}}(1|y)p_{\tilde{y}}(y)}{\sum_{y \in \{Spam, not\ \ spam\}} \prod_{i=1,4} p_{\tilde{x}[i]|\tilde{y}}(0|y)p_{\tilde{y}}(y) \prod_{i=2,3} p_{\tilde{x}[i]|\tilde{y}}(1|y)p_{\tilde{y}}(y)} \\
&= \underset{y \in \{Spam, not\ \ spam\}}{\arg\max} \{\frac{\frac{1}{2} \times \frac{1}{5} \times \frac{4}{5} \times \frac{2}{5} \times \frac{2}{5}}{\frac{1}{2} \times \frac{1}{5} \times \frac{4}{5} \times \frac{2}{5} \times \frac{2}{5} + \frac{1}{2} \times \frac{4}{5} \times \frac{1}{5} \times \frac{2}{5} \times \frac{3}{5}} \\
&, \frac{\frac{1}{2} \times \frac{4}{5} \times \frac{1}{5} \times \frac{2}{5} \times \frac{3}{5}}{\frac{1}{2} \times \frac{1}{5} \times \frac{4}{5} \times \frac{2}{5} \times \frac{2}{5} + \frac{1}{2} \times \frac{4}{5} \times \frac{1}{5} \times \frac{2}{5} \times \frac{3}{5}}\} \\
&= not\ spam
\end{aligned}
$$

.

We notice that this sentence is clealy spam since it contains provocative and monetary keyword but our naive bayes classifies it as "not spam". The shorting-coming turns out to arise from the assumptions that we made in the model, which is conditional independence given the hypothesis. Since sometimes the words in a sentence can only make practical sense when they come in pairs or trigrams, the assumption basically interpret each word separately. Moreover, the naive bayes relies heavily on the quality of dataset, where since we only have four indicators, which cannot fully encompass the suspicious keywords that have higher probablity in spam (for example, "send", "money").

2. (The Markov property) Let $\tilde{a}_1$, $\tilde{a}_2$, ..., $\tilde{a}_n$ be a Markov chain, where for any $1 < i < n$, $\tilde{a}_{i+1}$ is conditionally independent of $\tilde{a}_1, \ldots, \tilde{a}_{i-1}$ given $\tilde{a}_i$, i.e.

$$p_{\tilde{a}_{i+1}\,|\,\tilde{a}_1,\ldots,\tilde{a}_i}\left(a_{i+1}\,|\,a_1, a_2, \ldots, a_i\right) = p_{\tilde{a}_{i+1}\,|\,\tilde{a}_i}\left(a_{i+1}\,|\,a_i\right), \tag{1}$$

for any values of a_1, a_2, ..., a_n. Show that this implies that the future is conditionally independent from the past given the present:

$$p_{\tilde{a}_{i+1},\tilde{a}_{i-1}\,|\,\tilde{a}_i}\left(a_{i+1}, a_{i-1}\,|\,a_i\right) = p_{\tilde{a}_{i+1}\,|\,\tilde{a}_i}\left(a_{i+1}\,|\,a_i\right) p_{\tilde{a}_{i-1}\,|\,\tilde{a}_i}\left(a_{i-1}\,|\,a_i\right), \tag{2}$$

for any $2 \le i \le n-1$ and any values of a_{i-1}, a_i, a_{i+1}. (Hint: First show that $p_{\tilde{a}_{i+1}\,|\,\tilde{a}_{i-1},\tilde{a}_i}\left(a_{i+1}\,|\,a_{i-1}, a_i\right) = p_{\tilde{a}_{i+1}\,|\,\tilde{a}_i}\left(a_{i+1}\,|\,a_i\right)$ for a_{i-1}, a_i, a_{i+1}.)

We first show the probability in the hint and try to generalize it:

Proof. We know from the bayes theorem that:

$$\begin{aligned}
p_{\tilde{a}_{i+1}|\tilde{a}_{i-1},\tilde{a}_i}(a_{i+1|a_{i-1},a_i}) &= \frac{p_{\tilde{a}_{i+1},\tilde{a}_{i-1},\tilde{a}_i}(a_{i+1}, a_{i-1}, a_i)}{p_{\tilde{a}_{i-1},\tilde{a}_i}(a_{i-1}, a_i)} && \text{(Bayes theorem)} \\
&= \frac{\sum_{a_1,a_2,\cdots,a_{i-2}} p_{\tilde{a}_1}(a_1) \prod_{n=1}^{i-1} p_{\tilde{a}_{n+1}|\tilde{a}_n}(a_{n+1}|a_n) \cdot p_{\tilde{a}_{i+1}|\tilde{a}_i}(a_{i+1}|a_i)}{\sum_{a_1,a_2,\cdots,a_{i-2}} p_{\tilde{a}_1}(a_1) \prod_{n=1}^{i-1} p_{\tilde{a}_{n+1}|\tilde{a}_n}(a_{n+1}|a_n)} && \text{(Chain Rule)} \\
&= p_{\tilde{a}_{i+1}|\tilde{a}_i}(a_{i+1}|a_i)
\end{aligned}$$

Generally speaking, we could have $p_{\tilde{a}_{i+1}\,|\,\tilde{a}_{i-1},\tilde{a}_i}\left(a_{i+1}\,|\,a_{i-k}, \cdots, a_{i-1}, a_i\right) = p_{\tilde{a}_{i+1}\,|\,\tilde{a}_i}\left(a_{i+1}\,|\,a_i\right)$ for $a_{i-k}, \cdots, a_{i-1}$, a_i, a_{i+1} (where $i - k \ge 1$), which means that if we know the present, then the past information doesn't matter. □

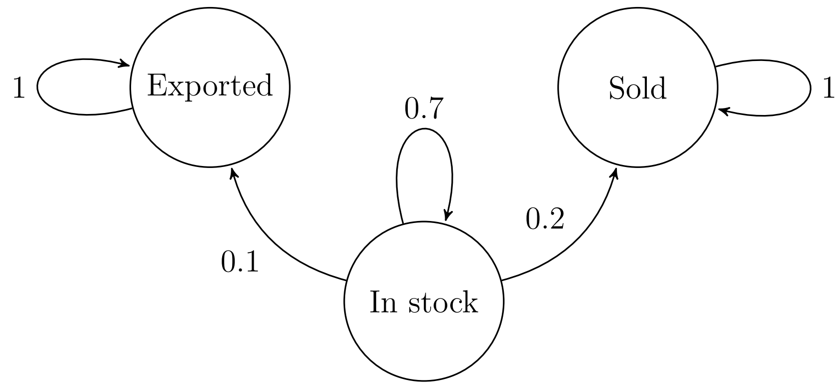
Once we have this, we can proceed with our proof as follows:

Proof.

$$\begin{aligned}
p_{\tilde{a}_{i+1},\tilde{a}_{i-1}|\tilde{a}_i}(a_{i+1}, a_{i-1}|a_i) &= \frac{p_{\tilde{a}_{i+1}|\tilde{a}_i,\tilde{a}_{i-1}}(a_{i+1}|a_i, a_{i-1}) p_{\tilde{a}_i,\tilde{a}_{i-1}}(a_{i-1}|a_i) p_{\tilde{a}_i}(a_i)}{p_{\tilde{a}_i}(a_i)} && \text{(Bayes Rule, Chain Rule)} \\
&= \frac{p_{\tilde{a}_{i+1}|\tilde{a}_i}(a_{i+1}|a_i) p_{\tilde{a}_i,\tilde{a}_{i-1}}(a_{i-1}|a_i) p_{\tilde{a}_i}(a_i)}{p_{\tilde{a}_i}(a_i)} && \text{(Using former lemma)} \\
&= p_{\tilde{a}_{i+1}|\tilde{a}_i}(a_{i+1}|a_i) p_{\tilde{a}_i,\tilde{a}_{i-1}}(a_{i-1}|a_i) && \text{(Arithmetic Operations)}
\end{aligned}$$

□

3. (Mobile phones) A company that makes mobile phones wants to model the sales of a new model they have just released. At the moment 90% of the phones are in stock, 10% have been sold locally and none have been exported. Based on past data, the company determines that each day a phone is sold with probability 0.2 and exported with probability 0.1. We define the following time-homogeneous Markov chain with three states to model this:



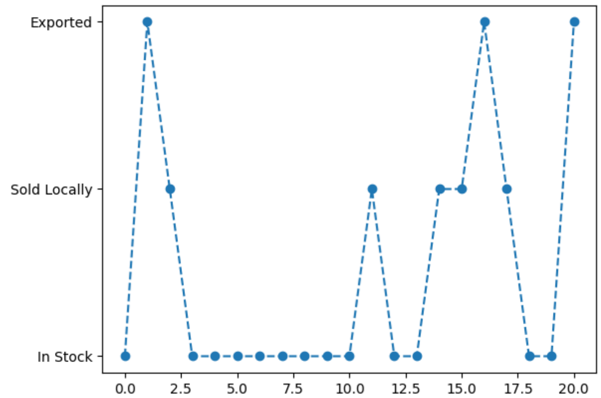
(a) What is the limit of the state vector π_i as $i \to \infty$?

We denote the state vector at the beginning to be $\pi_1 = \begin{bmatrix} 0.9 \\ 0.1 \\ 0 \end{bmatrix}$ and the transition matrix to be $T = \begin{bmatrix} 0.7 & 0 & 0 \\ 0.2 & 1 & 0 \\ 0.1 & 0 & 1 \end{bmatrix}$. We do a spectrum decomposition and could get $T = Q\Lambda Q^{-1} = \begin{bmatrix} 3/\sqrt{14} & 0 & 0 \\ -2/\sqrt{14} & 1 & 0 \\ -1/\sqrt{14} & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.7 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \sqrt{14}/3 & 0 & 0 \\ 2/3 & 1 & 0 \\ 1/3 & 0 & 1 \end{bmatrix}$.
We then apply the following:

$$
\begin{aligned}
\lim_{i\to\infty} \pi_i = T^{i-1}\pi_1 &= \begin{bmatrix} 3/\sqrt{14} & 0 & 0 \\ -2/\sqrt{14} & 1 & 0 \\ -1/\sqrt{14} & 0 & 1 \end{bmatrix} \lim_{i\to\infty} \begin{bmatrix} 0.7^{i-1} & 0 & 0 \\ 0 & 1^{i-1} & 0 \\ 0 & 0 & 1^{i-1} \end{bmatrix} \begin{bmatrix} \sqrt{14}/3 & 0 & 0 \\ 2/3 & 1 & 0 \\ 1/3 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.9 \\ 0.1 \\ 0 \end{bmatrix} \\
&= \left(\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} 2/3 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1/3 & 0 & 1 \end{bmatrix} \right) \begin{bmatrix} 0.9 \\ 0.1 \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ 0.7 \\ 0.3 \end{bmatrix}
\end{aligned}
$$

(b) Simulate the Markov chain and plot the evolution of the state vector.

Attched is the link to the code(click it): Jupyter notebook pdf link




```python
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# Problem 3(a): Limit of State Vector
T = np.array([[0.7,0,0],[0.2,1,0],[0.1,0,1]])
eigenvalues, eigenvectors = np.linalg.eig(T)
Q = eigenvectors
Q_inv = np.linalg.inv(eigenvectors)

# Problem 3(b): Simulate the Markov Chain
x0 = np.array([0.9,0.1,0])
T = np.array([[0.7,0,0],[0.2,1,0],[0.1,0,1]])
state_name = ["In Stock", "Sold Locally", "Exported"]


def one_step(state, transition_matrix):
    return np.random.choice(3, 1, p = transition_matrix[:, state].flatten())[0]


def simulate_discrete_markov_chain(start, transition_matrix, step):
    trace = [start]
    for i in range(step):
        trace.append(one_step(start, transition_matrix))

    return trace

def plot_trace(state_name, start_prob, transition_matrix, step):
    start_state = np.random.choice(len(state_name), 1, p = start_prob)[0]
    trace = simulate_discrete_markov_chain(start_state, transition_matrix, step)
    print(trace)
    plt.plot(np.arange(0, step + 1), trace, "o--")
    plt.yticks([0,1,2], state_name)

    plt.show()

plot_trace(state_name, x0, T, 20)
```

4. (Smoothing) In the application of naive Bayes, instead of computing the conditional pmf $p_{\tilde{x}[i]\,|\,\tilde{y}}(x[i]|y)$ empirically, we usually compute the conditional pmf as follows:

$$p_{\tilde{x}[i]\,|\,\tilde{y}}(\tilde{x}[i]\,|\,y) = \frac{\text{number of samples with label } y \text{ and having feature } x[i] + m}{\text{number of samples with label } y + md},$$

where m is a smoothing constant, and d is the number of features.

(a) What is the problem with naive Bayes when there is no smoothing ($m = 0$) and how does smoothing alleviate this?

When there is no smoothing, if the number of samples with label y could be zero, which results in zero division error when writing code.

(b) Complete the notebook *spam.ipynb*. Which m results in the highest accuracy on the test set?

It seems that when there is no smoothing, the prediction accuracy is maximized, and the bigger the smoothing constant is, the lower the accuracy is. The maximum accuracy occurs when m = 0, where zero division error happens. The parameter table is listed below for your reference:

m	accuracy
0	0.932
0.01	0.927
0.1	0.924
1	0.922
10	0.905
100	0.88
1000	0.845

Attched is the link to the code(click it): Jupyter notebook pdf link

(c) In the code, we compare the joint pmf $p_{\tilde{y},\tilde{x}[1],\cdots,\tilde{x}[d]}(y,\tilde{x}[1],\cdots\tilde{x}[d])$ for $y \in \{0,1\}$. Why is this equivalent to Equation (4.146) in the notes? Why do we need to apply the logarithm in practice?

Since ultimately we are comparing $p_{\tilde{y}\big|\tilde{x}[1],\cdots,\tilde{x}[d]}(y,\tilde{x}[1],\cdots\tilde{x}[d])$ for $y \in \{0,1\}$ and that $p_{\tilde{y}\big|\tilde{x}[1],\cdots,\tilde{x}[d]}(y,\tilde{x}[1],\cdots\tilde{x}[d])= \frac{p_{\tilde{y},\tilde{x}[1],\cdots,\tilde{x}[d]}(y,\tilde{x}[1],\cdots\tilde{x}[d])}{p_{\tilde{x}[1],\cdots,\tilde{x}[d]}(\tilde{x}[1],\cdots,\tilde{x}[d])}$ by bayes formula(conditional probability formula). Since the denominator is teh same across $y = 0,1$, we have that it's enough just to compare $p_{\tilde{y}\big|\tilde{x}[1],\cdots,\tilde{x}[d]}(y,\tilde{x}[1],\cdots\tilde{x}[d])$.

Since floating multiplication takes a huge amount of time, typically 5 clock cycles per elements whereas the floating addition only takes around 3 clock cycles per element. In other words, it is faster. Moreover, logarithm functions will preserve monotonicity of functions, which means $max\{a,b\} = max\{log(a),log(b)\}$, so applying it won't change the MAP result.

```
import numpy as np
from collections import Counter

def data_loader():
```

```
    # load data
    with open('train','r') as file:
        train_data = file.read().split('\n')[:-1]
    with open('test','r') as file:
        test_data = file.read().split('\n')[:-1]
    return train_data, test_data

def parser(datum):
    # extract labels and words
    email_addr, label, words = datum.split(' ',2)
    words = words.split()
    # transform words into dictionary
    word_dict = dict(zip([words[i] for i in
    range(0, len(words), 2)],
    [int(words[i+1]) for i in range(0, len(words), 2)]))
    # transform label into 0, 1
    if label == 'ham':
        label = 0
    elif label == 'spam':
        label = 1
    else:
        raise ValueError
    return label, word_dict

def data_preprocessing(train_data, test_data):
    y_train = np.zeros(len(train_data))
    y_test = np.zeros(len(test_data))
    x_train = []
    x_test = []
    for i, datum in enumerate(train_data):
        label, word_dict = parser(datum)
        y_train[i] = label
        x_train.append(word_dict)
    for i, datum in enumerate(test_data):
        label, word_dict = parser(datum)
        y_test[i] = label
        x_test.append(word_dict)
    return x_train, y_train, x_test, y_test

def compute_empirical_pmf_y(y_train):
    # compte distribution P(y=1), P(y=0)
    # TODO
    return np.sum(y_train == 1) / len(y_train),  np.sum(y_train == 0) / len(y_train)

def m_estimation_conditional_probability(x_train_frt, y_train, num_vocab, m):
    # compute P(x_j|y=1) and P(x_j|y=0) for j = 1, ..., d
    # TODO
    p_on_spam = np.zeros((2, x_train_frt.shape[1]))
    p_on_ham = np.zeros((2, x_train_frt.shape[1]))

    temp_df = pd.DataFrame(np.hstack([x_train_frt,y_train.reshape(-1,1)]))
    temp_df.columns = temp_df.columns = [*list(temp_df.columns)[:-1],"label"]

    for j in range(num_vocab):
        column_data_0 = (temp_df[temp_df["label"]==0])[j]
        column_data_1 = (temp_df[temp_df["label"]==1])[j]

        # P(x_j = 1|y=0) by definition
        p_on_ham[1, j] = (sum(column_data_0 > 0) + m) / (len(column_data_0) + m * num_vocab)
        # P(x_j = 0|y=0) by the fact that conditional probability is a valid pmf
        p_on_ham[0, j] = 1 - p_on_ham[1, j]



        # P(x_j = 1|y=1)
        p_on_spam[1, j] = (sum(column_data_1 > 0) + m) / (len(column_data_1) + m * num_vocab)
        # P(x_j = 0|y=1)
        p_on_spam[0, j] = 1 - p_on_spam[1, j]

    return p_on_spam, p_on_ham


def log_estimated_probability(p_spam, p_ham, p_on_spam_m, p_on_ham_m, x_frts):
    # compute log(P(y=1, x_1, x_2,..., x_n)) and log(P(y=0, x_1, x_2,..., x_n))
    # hint: log(P(y, x_1, x_2,..., x_n)) = log(P(y=1)) + log(P(x_1 | y)) + ... + log(P(x_d | y))
    # TODO

    log_p_spam, log_p_ham = np.zeros((len(x_frts), )), np.zeros((len(x_frts), ))

    for i in range(len(x_frts)):
        log_p_spam[i] = np.sum([np.log(p_spam)]
        +[np.log(p_on_spam_m[1 if x_frts[i, j] > 0 else 0 ,j]) for j in range(x_frts.shape[1])])
        log_p_ham[i] = np.sum([np.log(p_ham)]
        +[np.log(p_on_ham_m[1 if x_frts[i, j] > 0 else 0 ,j]) for j in range(x_frts.shape[1])])

    return log_p_spam,log_p_ham

def accuarcy(y_true, y_pred):
    # calculate accuracy
    # TODO

    return np.sum(y_true == y_pred) / len(y_true)
```

```python

from sklearn.feature_extraction import DictVectorizer
# load data
train_data, test_data = data_loader()

# extract labels to 0,1 and features to dicticnary
x_train, y_train, x_test, y_test = data_preprocessing(train_data, test_data)

# transform word dicts to feature vectors
vectorizer = DictVectorizer(sparse=False)
x_train_frt = vectorizer.fit_transform(x_train)
x_test_frt = vectorizer.transform(x_test)
print(x_train_frt.shape, x_test_frt.shape)

def pipeline(x_train_frt, y_train, x_test_frt, y_test, m):
    p_spam, p_ham = compute_empirical_pmf_y(y_train)
    p_on_spam_m, p_on_ham_m = m_estimation_conditional_probability(x_train_frt, y_train, x_train_frt.shape[1]
    log_p_spam, log_p_ham = log_estimated_probability(p_spam, p_ham, p_on_spam_m, p_on_ham_m, x_test_frt)
    test_pred = (log_p_spam > log_p_ham)
    print(str(m) + ":" + str(accuarcy(y_test, test_pred)))

m_grid = [0, 0.01, 0.1, 1, 10, 100, 1000]
for m in m_grid:
    pipeline(x_train_frt, y_train, x_test_frt, y_test, m)
```