Name: Jianyuan Lu GT ID: 903633385

1.Basic Optimization

(a)

Since the log likelihood function is:

$$l(\theta) = \sum_{i=1}^{m} \{-\log(1+\exp\{-\theta x^i\}) + (y^i - 1)\theta x^i\}$$

We want to get the $\hat{\theta}$ such that $l(\hat{\theta})$ gets the maximum value, so we will take the derivative of $l(\theta)$ with respective to θ.

$$\begin{aligned}
\frac{\partial l(\theta)}{\partial \theta} &= \sum_{i=1}^{m} \frac{\partial\{-\log(1+\exp\{-\theta x^i\}) + (y^i - 1)\theta x^i\}}{\partial \theta} \\
&= \sum_{i=1}^{m} (y^i - 1)x^i + \frac{\partial\{-\log(1+\exp\{-\theta x^i\})\}}{\partial \theta} \\
&= \sum_{i=1}^{m} (y^i - 1)x^i - \left\{\frac{1}{1+\exp(-\theta x^i)} \times \frac{\partial \exp(-\theta x^i)}{\partial \theta}\right\} \\
&= \sum_{i=1}^{m} (y^i - 1)x^i - \left\{\frac{1}{1+\exp(-\theta x^i)} \times \exp(-\theta x^i) \times (-x^i)\right\} \\
&= \sum_{i=1}^{m} (y^i - 1)x^i + \frac{\exp(-\theta x^i)x^i}{1+\exp(-\theta x^i)}
\end{aligned}$$

(b)

```
float GradientAscent(array[] x,array[] y){ #function to use the GradientAscent algorithm

    float calGraident(int theta){  #compute the gradient at any point theta
        float gradientInitial = 0;
        for(int i =0;i<x.size;i++){
        gradientInitial = gradientInitial + (y[i]-1)*x[i]+exp(-theta*x[i])*x[i]/(1+exp(-theta*x[i]));
        }
        return gradientInitial;
    }
    int t0 = np.random.randint(0, 3)  #choose a random point
    gradient0 = calGraident(t0);

    while(abs(gradient0)>0.01){  #go along the gradient direction untill it converge
        int j = 1;  #set the step
        step = 1/j;
        t0 = t0 + step*gradient0;
        gradient0 = calGraident(t0);
    }
    return t0;
}
```

(c)

```
StoGradient(x,y){


    index = [] //form the index list
    for i in range(len(x)):
        index.append(i)

    shuffle(index) //shuffle the index list
    K = 5; //the number of subset
    m = int(len(index)/K)
    list2 = []
    //randomly split the dataset into K subset
    for i in range(0, len(list), m):
        indexNew.append(list[i:i+m])


    def calGraident(theta,subIndex){    //#compute the gradient at any point theta using the data subset
        gradientInitial = 0;
        for(int i =0;i<len(subIndex);i++){
        gradientInitial = gradientInitial + (y[subIndex[i]]-1)*x[subIndex[i]]\
        +exp(-theta*x[subIndex[i]])*x[subIndex[i]]/(1+exp(-theta*x[subIndex[i]]));
        }
        return gradientInitial;
    }

    t0 = np.random.randint(0, 3)
    gradient0 = calGraident(t0);

    while(abs(gradient0)>0.01){
        j = 1;
        step = 1/j;
        subIndex = indexNew[(j-1)%5] // iterately using the subsets
        t0 = t0 + step*gradient0;
        gradient0 = calGraident(t0,subIndex);
    }
    return t0;
}
```

(d)

Since θ is one dimensional, so the hessian matrix becomes to a scalar.

$$\frac{\partial^2 l(\theta)}{\partial \theta^2} = \sum_{i=1}^{m} \frac{\partial \left(y^i - 1\right)x^i + \frac{\exp\left(-\theta x^i\right)x^i}{1+\exp(-\theta x^i)}}{\partial \theta}$$

$$= \sum_{i=1}^{m} \frac{\exp\left(-\theta x^i\right)x^i\left(-x^i\right)\left[1+\exp\left(-\theta x^i\right)\right] - \exp\left(-\theta x^i\right)x^i\exp\left(-\theta x^i\right)\left(-x^i\right)}{\left[1+\exp(-\theta x^i)\right]^2}$$

$$= \sum_{i=1}^{m} \frac{-\left(x^i\right)^2\exp\left(-\theta x^i\right)}{\left[1+\exp(-\theta x^i)\right]^2}$$

So we can find that since $\frac{\partial^2 l(\theta)}{\partial \theta^2}$ is always negative, so the function $l(\theta)$ is concave. As we know, the gradient descent method will get the local maximum value. Since $l(\theta)$ is concave, so the local maximum will become global maximum, which makes the gradient descent achieve a unique global optimizer.

2.Comparing Classifiers

Part one

(a)

The testing accuracy may vary according to the number of train samples. The result is show as below:

For the K Nearest Neighbors:

Training with 80% samples		
K Nearest Neighbors		
the training accuracy: 0.9853		
	predicted 0	predicted 1
true 0	67	0
true 1	2	67

Testing with 20% samples		
K Nearest Neighbors		
the testing accuracy: 0. 9412		
	predicted 0	predicted 1
true 0	19	0
true 1	2	13

For the Naive Bayes:

Training with 80% samples		
Naive Bayes		
the training accuracy: 0.9853		
	predicted 0	predicted 1
true 0	67	0
true 1	2	67

Testing with 20% samples		
Naive Bayes		
the testing accuracy: 0. 9412		
	predicted 0	predicted 1
true 0	19	0
true 1	2	13

For the LogisticRegression:

Training with 80% samples		
LogisticRegression		
the training accuracy: 1.0		
	predicted 0	predicted 1
true 0	67	0
true 1	0	69

Testing with 20% samples		
LogisticRegression		
the testing accuracy: 0. 9412		
	predicted 0	predicted 1
true 0	19	0
true 1	2	13

From the result above, we could find that the LogisticRegression behaves a little bit better than other two method. This is especially true when the training data is small and testing data is large. The following graph can prove this.

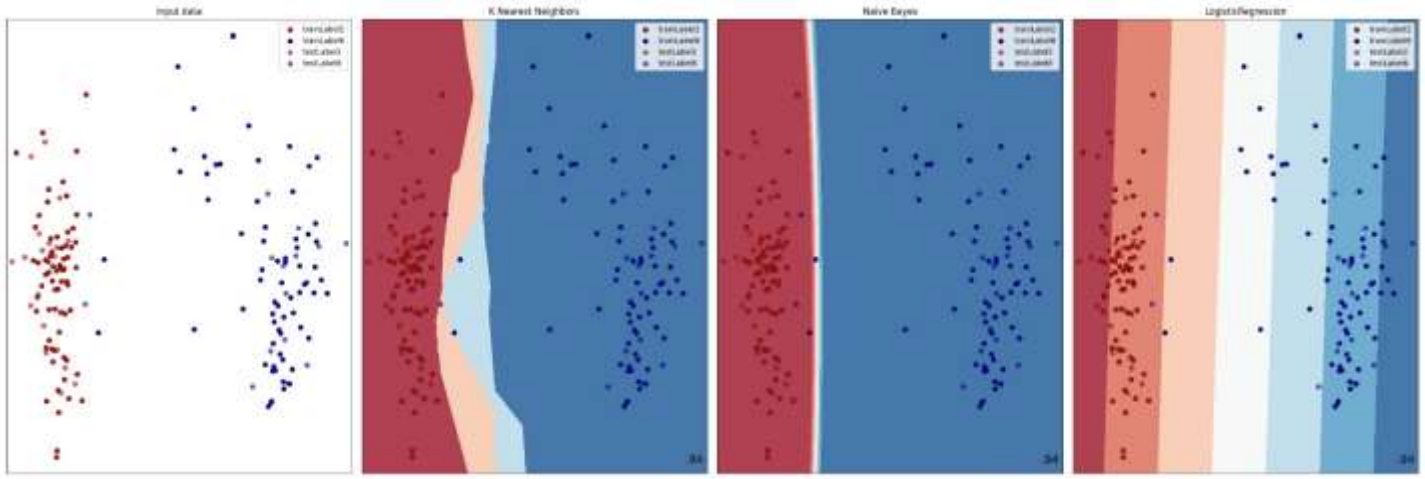
Testing with 98% samples and training with 2% samples		
	Training accuracy	Testing accuracy
K Nearest Neighbors	0.6667	0.491
Naive Bayes	1.0	0.491
LogisticRegression	1	0.976

I think the result why LogisticRegression behaves better is that it takes the least assumptions and use all the information from the dataset. For K Nearest Neighbors, although it use no parameters, but it depends heavily on the training dataset. So for small dataset, it behaves bad. For Naive Bayes, it takes too much assumptions. If the dataset's likelihood function is not Gauss distribution, it will behave bad. Besides, estimating the co-variance matrix needs many samples, so it will behave bad for small dataset.

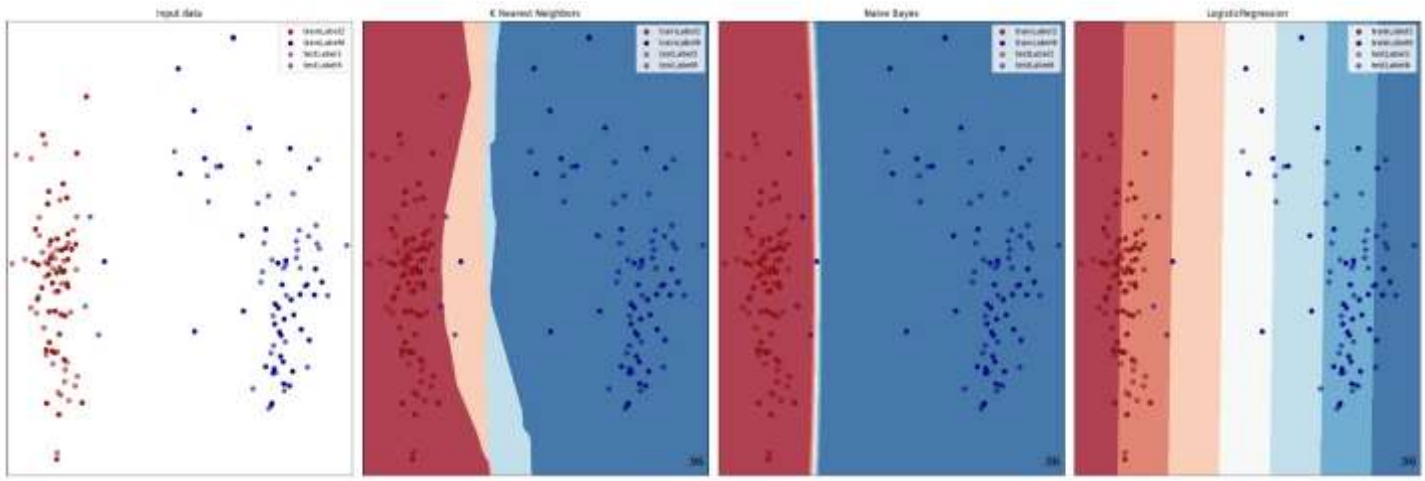
(b).

I draw the decision boundary for three different classifiers in three different test sample proportions respectively. The graph is shown as below:

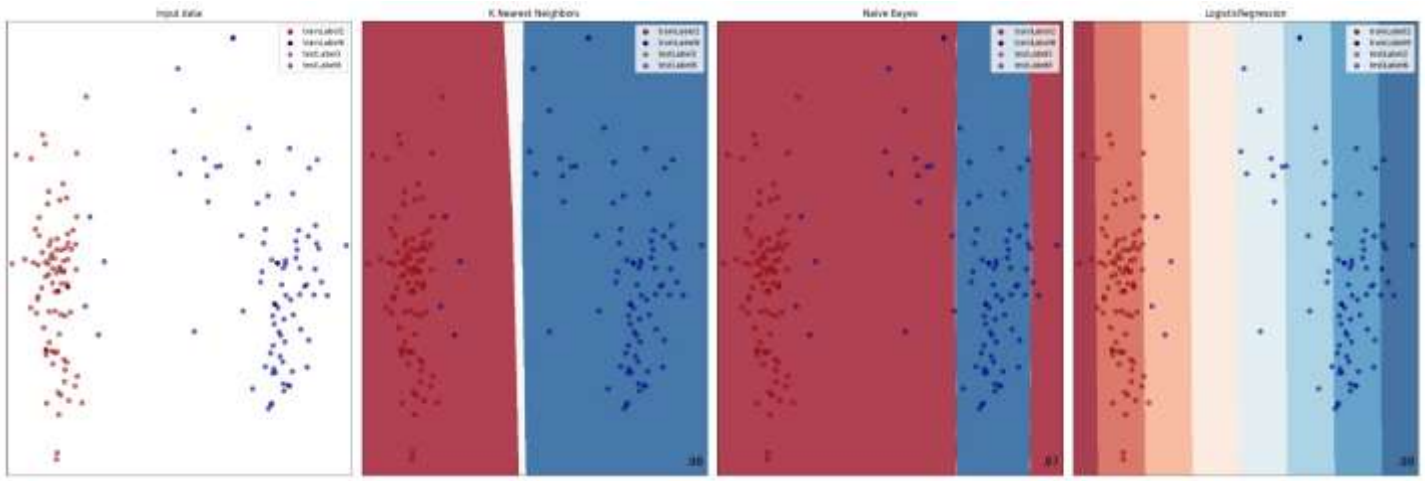
Test sample = 20%



Test sample = 50%



Test sample = 97%



For the LogisticRegression classifier, since it just uses the linear mapping θx^i, so the width of each contour area is evenly divided, and the boundary is relatively smooth and becomes a line. For the Naive Bayes classifier, the width of each contour area is not evenly divided. The areas for 0 or 1 is vary large compared with other values. That is because Naive Bayes uses Gaussian distributions and the likelihood value is very high when the point is closed to the mean. For the K Nearest Neighbors classifier, we could identify that the boundary is really noisy and jagged.

Part two

(a)

For the MNIST dataset, we could identify that the picture 0~1031 is label '2', and picture 1032~1989 is label '6'. Perform the three classifiers with testing sample proportion 20%.

For the K Nearest Neighbors:

Training with 80% samples		
K Nearest Neighbors		
the training accuracy: 0.9975		
	predicted 2	predicted 6
true 2	807	4
true 6	0	781

Testing with 20% samples		
K Nearest Neighbors		
the testing accuracy: 0. 9925		
	predicted 2	predicted 6
true 2	219	2
true 6	1	176

For the Naive Bayes:

Training with 80% samples		
Naive Bayes		
the training accuracy: 0.7412		
	predicted 2	predicted 6
true 2	399	412
true 6	0	781

Testing with 20% samples		
Naive Bayes		
the testing accuracy: 0.7337		
	predicted 2	predicted 6
true 2	118	103
true 6	3	174

For the LogisticRegression:

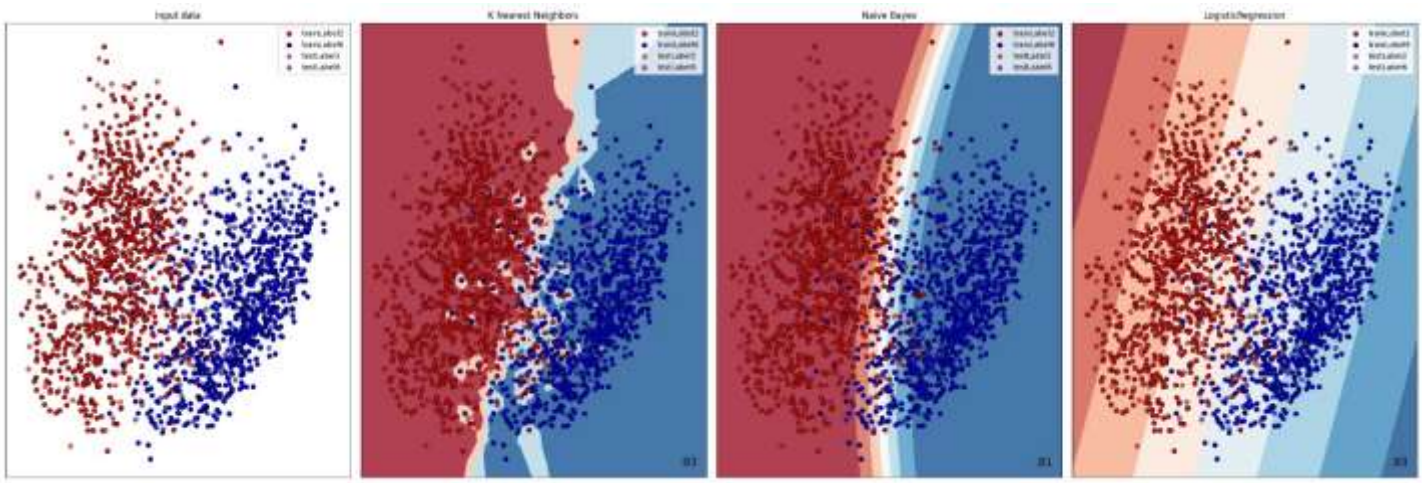
Training with 80% samples		
LogisticRegression		
the training accuracy: 0.9994		
	predicted 2	predicted 6
true 2	810	1
true 6	0	781

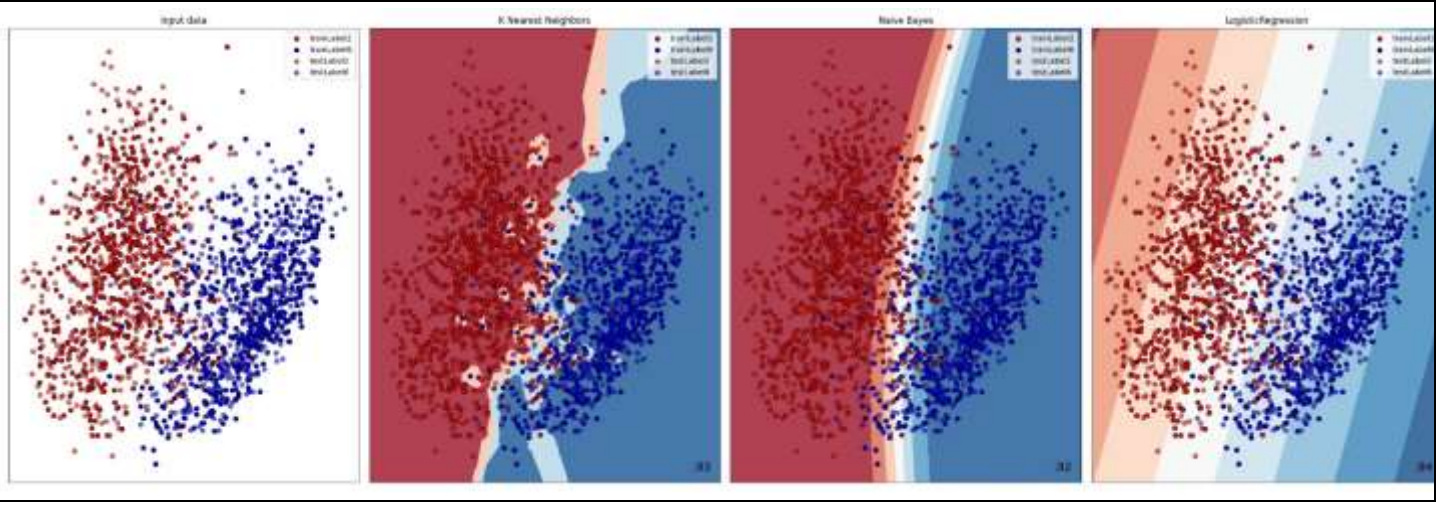
Testing with 20% samples		
LogisticRegression		
the testing accuracy: 0.9799		
	predicted 2	predicted 6
true 2	216	5
true 6	3	174

From the result above, we could find that the K Nearest Neighbors behaves better than LogisticRegression. And the behavior of Naive Bayes is poor. The reason is that the MNIST dataset is the Handwritten digits, which is the matrix showing the shape of label '2' and label '6'. Since the data purely shows the shape, there is no abstractive information. In this case, K Nearest Neighbors will behave better because it use the nearest training data to identify the test data, and the data showing the similar shape will be located closely.

(b).

I draw the decision boundary for three different classifiers in three different test sample proportions respectively. The graph is shown as below:

Test sample = 20%


Test sample = 50%


For the LogisticRegression classifier, since it just uses the linear mapping θx^i, so the width of each contour area is evenly divided, and the boundary is relatively smooth and becomes a line. For the Naive Bayes classifier, the width of each contour area is not evenly divided. The areas for 0 or 1 is vary large compared with other values. That is because Naive Bayes uses Gaussian distributions and the likelihood value is very high when the point is closed to the mean.

For the K Nearest Neighbors classifier, we could identify that the boundary is really noisy and jagged. Even there is some holes in the red area, that is because there is some blue points lie in the red area.

3.Spam filtering

(a).

For each message, the corresponding feature vector is:

Spam message	
Message content	Attribute vector
Million, dollar, offer	$\mathbf{x} = [0,1,0,0,0,0,0,1,1,0,0,0,0,0,0]^T$
Secret, offer, today	$\mathbf{x} = [1,1,0,0,0,0,1,0,0,0,0,0,0,0,0]^T$
Secret, is, secret	$\mathbf{x} = [2,0,0,0,0,0,0,0,0,0,1,0,0,0,0]^T$

Non-spam message	
Message content	Attribute vector
Low, price, for, valued, customer	$\mathbf{x} = [0,0,1,1,1,1,0,0,0,0,0,1,0,0,0]^T$
Play, secret, sports, today	$\mathbf{x} = [1,0,0,0,0,0,1,0,0,1,0,0,1,0,0]^T$
Sports, is, healthy	$\mathbf{x} = [0,0,0,0,0,0,0,0,0,1,1,0,0,1,0]^T$
Low, price, pizza	$\mathbf{x} = [0,0,1,1,0,0,0,0,0,0,0,0,0,0,1]^T$

So. $\mathrm{P}(y = 0) = \frac{3}{7}$, $\mathrm{P}(y = 1) = \frac{4}{7}$

(b).

$$l(\theta_{c,k}) = \sum_{i=1}^{m}\sum_{k=1}^{d} x_k^{(i)} log\theta_{y^{(i)},k}$$

Since $\sum_{c=0}^{1}\theta_{y^c,k} = 1$

So, by using Lagrange multiplier, we could have the following equation:

$$L(\theta_{c,k}) = \sum_{i=1}^{m}\sum_{k=1}^{d} x_k^{(i)} log\theta_{y^{(i)},k} + \lambda\left(1 - \sum_{c=0}^{1}\theta_{y^c,k}\right)$$

$$\frac{\partial L(\theta_{c,k})}{\partial\theta_{c,k}} = \sum_{i=1}^{m} x_k^{(i)}\frac{1}{\theta_{c,k}}\left(y^{(i)} == c\right) - \lambda$$

Let $\frac{\partial L(\theta_{c,k})}{\partial\theta_{c,k}}$=0 in order to get the $L(\theta_{c,k})$ to meet its maximum value, we have:

$$\lambda = \sum_{i=1}^{m} x_k^{(i)}\frac{1}{\theta_{c,k}}\left(y^{(i)} == c\right)$$

$$\theta_{c,k} = \frac{\sum_{i=1}^{m} x_k^{(i)}\left(y^{(i)} == c\right)}{\lambda}$$

As we know that $\sum_{c=0}^{1}\theta_{y^c,k} = 1$, so we could find the expression for λ:

$$\lambda = \sum_{i=1}^{m} x_k^{(i)}$$

So, we could get the expression for $\theta_{c,k}$:

$$\theta_{c,k} = \frac{\sum_{i=1}^{m} x_k^{(i)}\left(y^{(i)} == c\right)}{\sum_{i=1}^{m} x_k^{(i)}} \qquad c = 0,1; k = 0,1,\cdots,15$$

(c).

For "today is secret", the attribute vector $\mathbf{x} = [1,0,0,0,0,0,1,0,0,0,1,0,0,0,0]^T$

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

So, we could calculate $P(c = 0|x)$ and $P(c = 1|x)$ respectively.

By training data, we get the parameters, which are shown as below:

$$\theta_{0,1} = \frac{3}{4}$$

$$\theta_{0,7} = \frac{1}{2}$$

$$\theta_{0,11} = \frac{1}{2}$$

$$\theta_{1,1} = \frac{1}{4}$$

$$\theta_{1,7} = \frac{1}{2}$$

$$\theta_{1,11} = \frac{1}{2}$$

$$P(c=0|x) = \frac{P(x|c=0)P(c=0)}{P(x)}$$
$$= \frac{e^{\left(1\times\log\left(\frac{3}{4}\right)+1\times\log\left(\frac{1}{2}\right)+1\times\log\left(\frac{1}{2}\right)\right)} \times \frac{3}{7}}{P(x)}$$
$$= \frac{0.0804}{P(x)}$$

$$P(c=1|x) = \frac{P(x|c=1)P(c=1)}{P(x)}$$
$$= \frac{e^{\left(1\times\log\left(\frac{1}{4}\right)+1\times\log\left(\frac{1}{2}\right)+1\times\log\left(\frac{1}{2}\right)\right)} \times \frac{4}{7}}{P(x)}$$
$$= \frac{0.0357}{P(x)}$$

So, Since $P(c=0|x) > P(c=1|x)$, so it is spam.