Homework_1

Bauyrzhan Zhakanov

16 September 2019

1 Learning exercises

1.1 Exercise 1.1

1.1.1 Medical diagnosis

the input space X: medical history and symptoms
the output space Y: medical diagnosis of patients
the target function f: (X $\rightarrow$ Y): based the medical history and symptoms, finding the formula for identification of the diagnosis for the patients

1.1.2 Handwritten digit recognition

the input space X: pictures of hand digit
the output space Y: zip code recognition
the target function f:(X $\rightarrow$ Y): an algorithm that sorts mail using hand digit pictures

1.1.3 Spam Determination

the input space X: any emails
the output space Y: filters spam emails
the target function f:(X $\rightarrow$ Y): an algorithm that identifies email spam or not

1.1.4 Electric load problem

the input space X: price, temperature and day of the week
the output space Y: choosing the electric load
the target function f:(X $\rightarrow$ Y): predicting the variation of electric board using price, temperature, and day of the week

1.1.5 Data prediction solver

the input space X: past data
the output space Y: empirical solution
the target function f:(X $\rightarrow$ Y): an algorithm to make an empirical solution based on the past data

1.2 Exercise 1.5

1.2.1 Determining the age at which a particular medical test should be performed

learning approach

1.2.2 Classifying numbers into primes and non-primes

design approach

1.2.3 Detecting potential fraud in credit card charges

learning approach

1.2.4 Determining the time it would take a falling object to hit the ground

design approach

1.2.5 Determining the optimal cycle for traffic lights in a busy intersection

learning approach

2 Perceptron Learning Algorithm

2.1 Show that $y(t)w(t)x(t) < 0$

Output can hold only -1 and +1. If our weight have a negative sign or $x(t)$ is misclassified by $w(t)$, so it could be $-y(t) = -w(t)x(t) < 0$. This is telling that $-y(t)y(t) < 0$, so that $y(t)w(t)x(t) < 0$.

2.2 Show that $y(t)w^T(t+1)x(t) > y(t)w^T(t)x(t)$

If we decompose $w(t+1) = w(t) + y(t)x(t)$. So that,

$$y(t)w^T(t+1)x(t) > y(t)w^T(t)x(t)$$

$$y(t)(w(t) + y(t)x(t))^T x(t) > y(t)w^T(t)x(t)$$
$$y(t)w(t)^T x(t) + y^2(t)x^2(t) > y(t)w^T(t)x(t)$$

$$y^2(t)x^2(t)x(t) > 0$$

It shows that $y(t)w^T(t+1)x(t) > y(t)w^T(t)x(t)$.

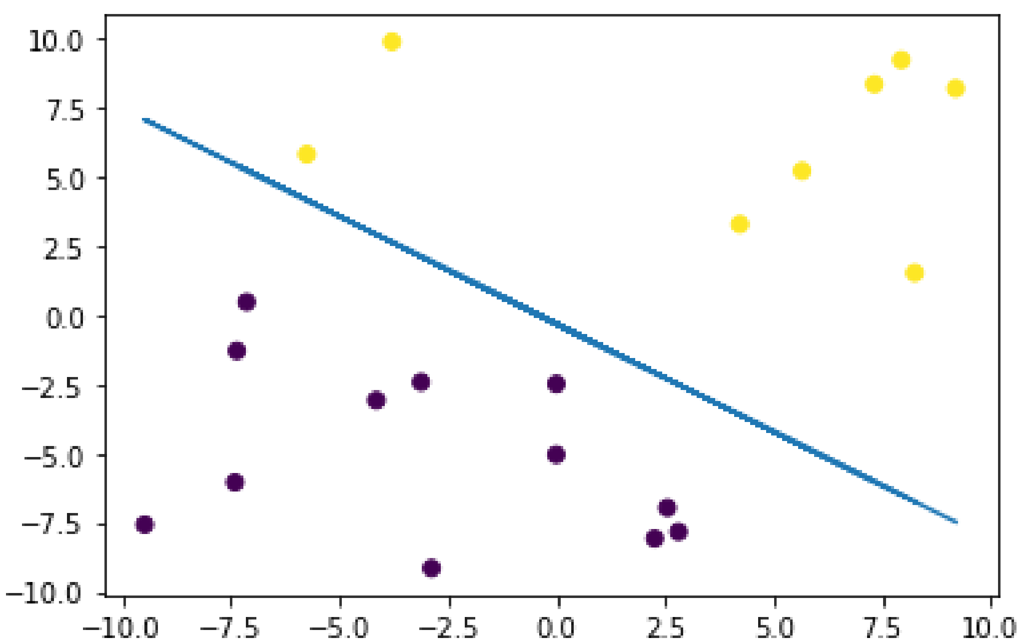
2.3 As far as classifying x(t) is concerned, argue that the move from $w(t)$ to $w(t+1)$ is a move 'in the right direction

In the case, when $w^T(t)$ is small, $w(t+1) = w(t) + w(0)x(t)$ must increase to the right. Otherwise, $w(t+1) = w(t) - w(0)x(t)$ must decrease to the right.

3 Experiments with Perceptron Learning Algorithm

In [113]:
```
import numpy as np
from matplotlib import pyplot as plt
```

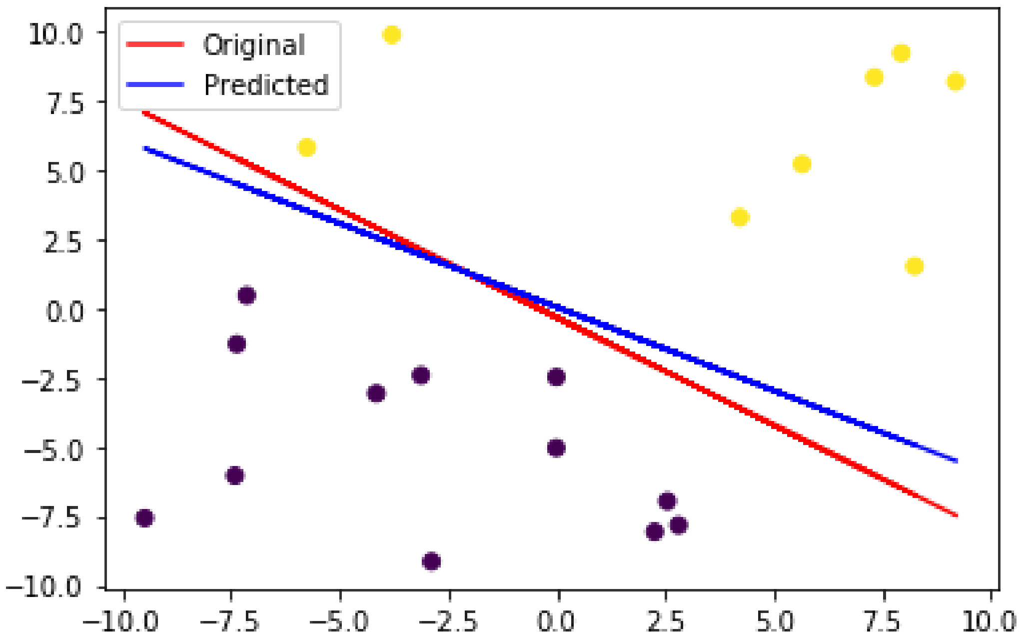
In [116]:
```
#To generate 20 dataset
d = 2 #dimension is 2
w = np.random.randint(1,10, d) #weights
b = np.random.randint(1,10)  #some b
x = np.random.uniform(-1,1,(20,d))*10 #our inputs
y = np.zeros(20)  #corresponding outputs
#some random line
h = x.dot(w) + b
#and labels
label = (h > 0)*1
y = label
plt.scatter(x[:, 0], x[:, 1], c=label)
plt.plot(x[:,0],-w[0]*x[:,0]/w[1]-b/w[1])
plt.show()
```



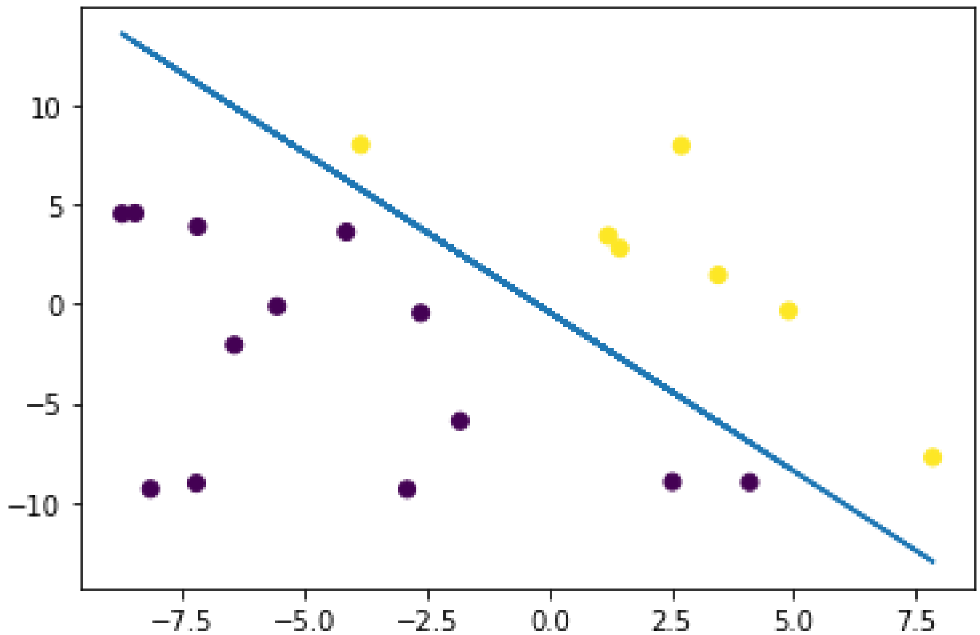
```
In [117]: class Perceptron:
              #initials
              def __init__(self, w, b, d):
                  self.w = w
                  self.b = b
                  self.d = d
              #prediction funct
              def predict(self, x_data):
                  activ = np.dot(x_data, self.w_t[0] + self.w_t[1:])
                  return activ
              #fitting values
              def fit(self, X, Y, size = 20, iters = 100):
                  self.w_t = np.zeros(self.d+1)
                  count = 0

                  #iterations
                  for count in range(iters):
                      count += 1
                      if count == iters:
                          break

                      #updating
                      for i in range(X.shape[0]):
                          prediction = self.predict(X[i])
                          training = Y[i]

                          if prediction == 0:
                              prediction = -1
                          if training == 0:
                              training = -1

                          if (prediction != training):
                              self.w_t[0] = self.w_t[0] + training
                              self.w_t[1:] = self.w_t[1:] + training*x[i]

#results
def results(self, X, Y):
    #plt.colors = ['g' if l == 0 else 'b' for l in Y]
    plt.scatter(X[:,0], X[:,1], c=Y)
    #plt.legend()
    plt.plot(X[:,0],-self.w[0]*X[:,0]/self.w[1]-self.b/self.w[1], color='r', label='Original')
    plt.plot(X[:,0],-self.w_t[1]*X[:,0]/self.w_t[2]-self.w_t[0]/self.w_t[2], color='b', label='P
    plt.legend()

    #Values
```

```
p = Perceptron(w, b, d)
print("Perceptron of original and predicted values")
p.fit(x, y)
p.results(x, y)
```

```
Perceptron of original and predicted values
```

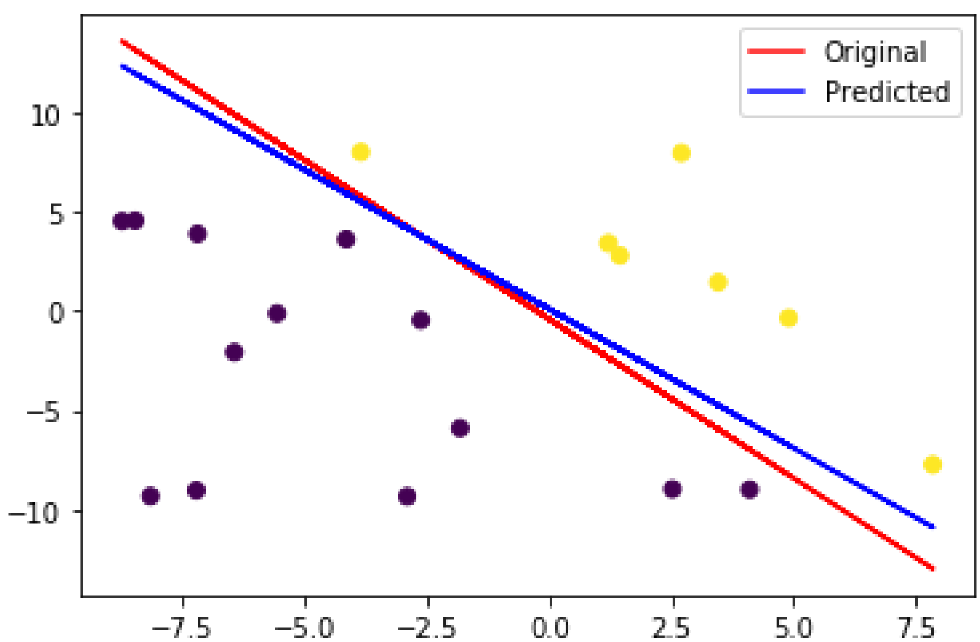


In [123]:
```
#To generate another 20 dataset
d = 2
w = np.random.randint(1,10, d) #weights
b = np.random.randint(1,10)  #some b
x = np.random.uniform(-1,1,(20,d))*10 #our inputs
y = np.zeros(20)  #corresponding outputs
#some random line
h = x.dot(w) + b
#and labels
label = (h > 0)*1
y = label
plt.scatter(x[:, 0], x[:, 1], c=label)
plt.plot(x[:,0],-w[0]*x[:,0]/w[1]-b/w[1])
plt.show()
```

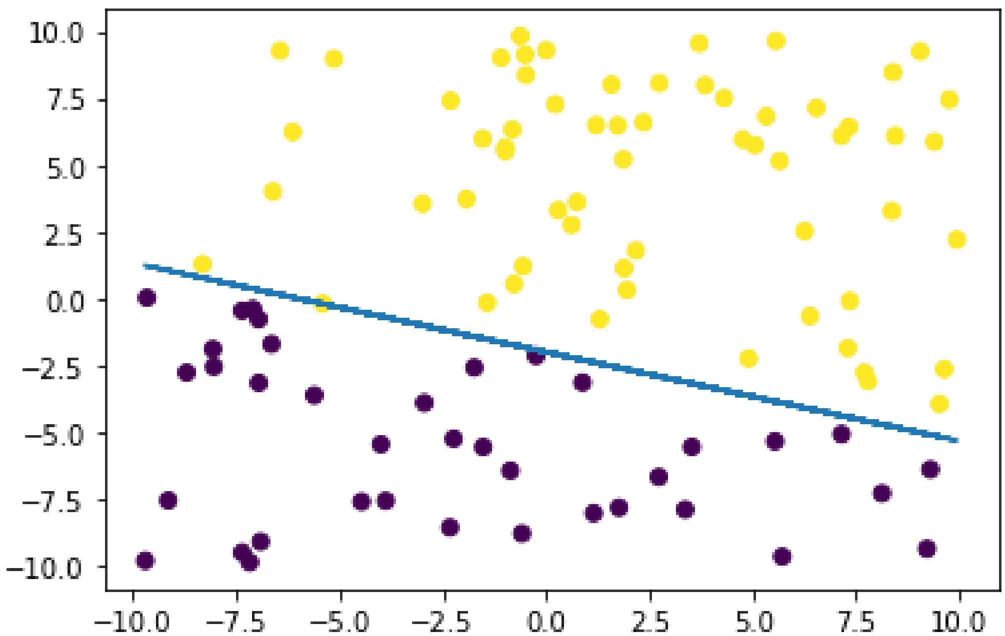



```
In [124]: p = Perceptron(w, b, d)
          print("Perceptron of original and predicted values")
          p.fit(x, y)
          p.results(x, y)
```

```
Perceptron of original and predicted values
```

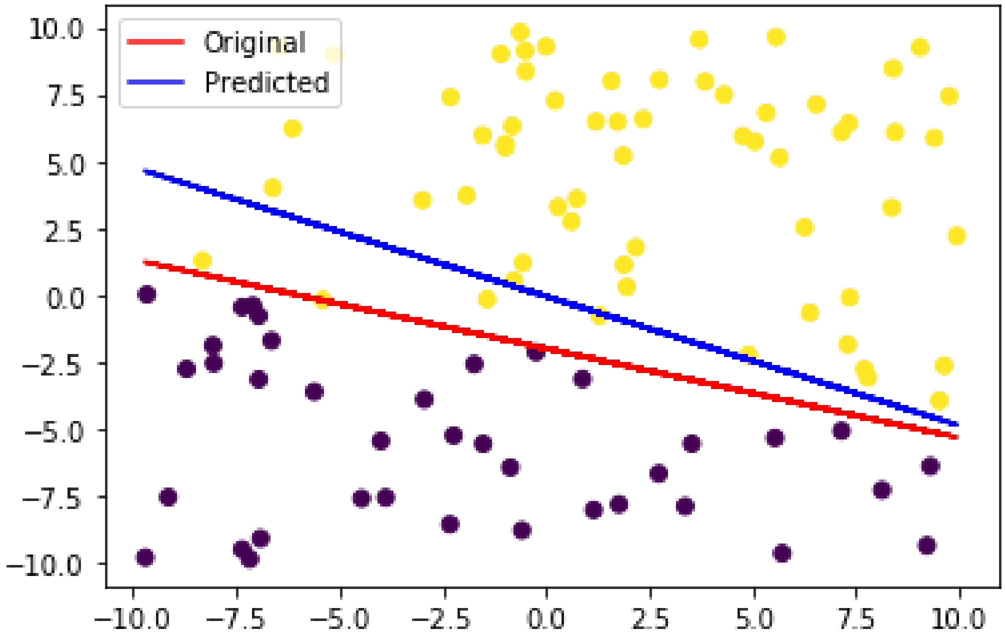


In [125]:
```
#To generate 100 dataset
d = 2
w = np.random.randint(1,10, d) #weights
b = np.random.randint(1,10)  #some b
x = np.random.uniform(-1,1,(100,d))*10 #our inputs is now 100
y = np.zeros(20)  #corresponding outputs
#some random line
h = x.dot(w) + b
#and labels
label = (h > 0)*1
y = label
plt.scatter(x[:, 0], x[:, 1], c=label)
plt.plot(x[:,0],-w[0]*x[:,0]/w[1]-b/w[1])
plt.show()
```

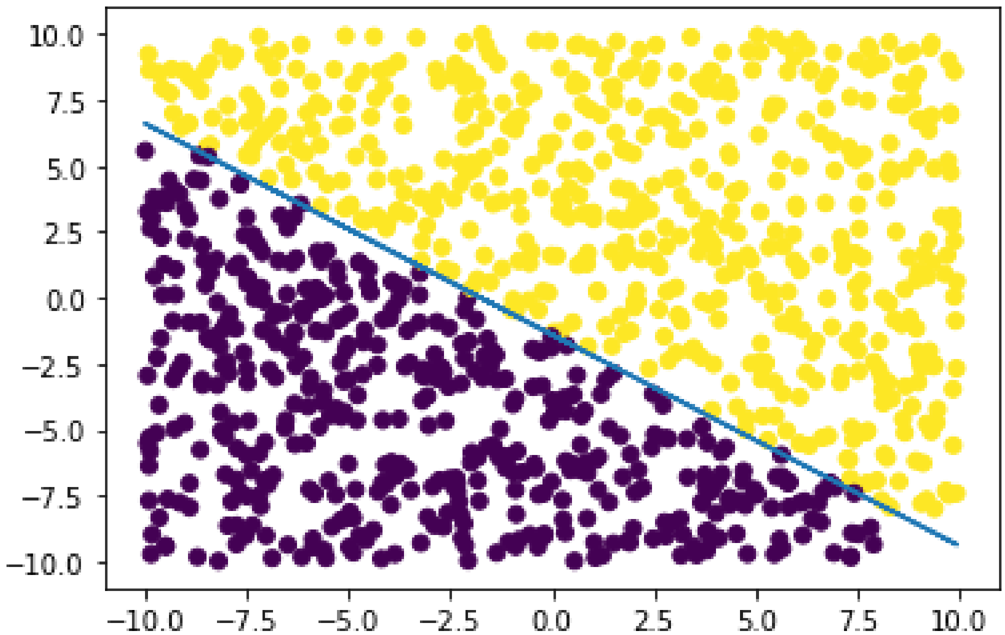


In [126]:
```
p = Perceptron(w, b, d)
print("Perceptron of original and predicted values")
p.fit(x, y, size = 100)
p.results(x, y)
```

```
Perceptron of original and predicted values
```

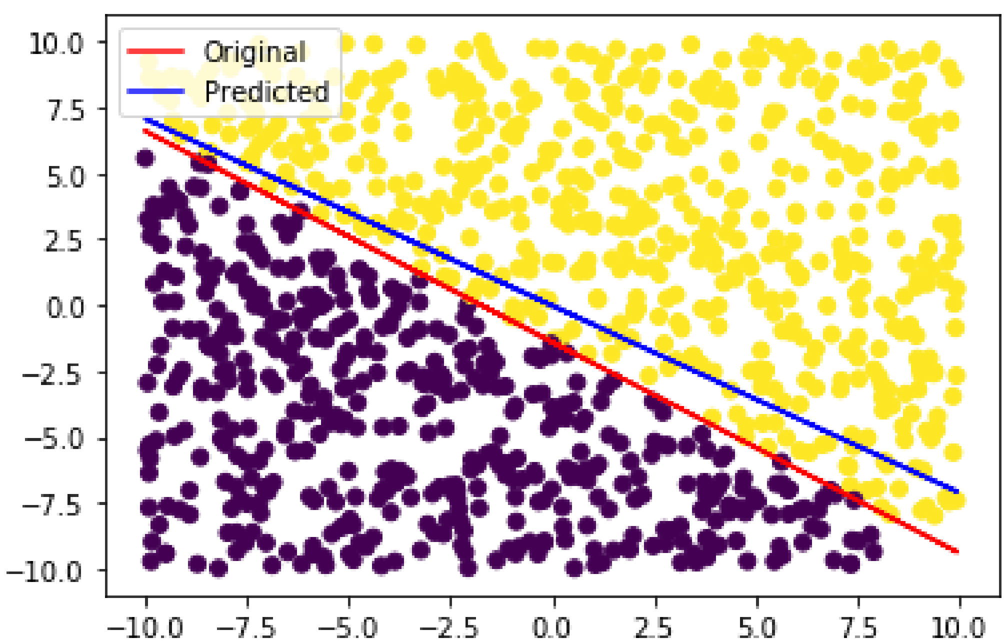



```
In [127]: #To generate 1000 dataset
          d = 2
          w = np.random.randint(1,10, d) #weights
          b = np.random.randint(1,10)   #some b
          x = np.random.uniform(-1,1,(1000,d))*10 #our inputs
          y = np.zeros(20)  #corresponding outputs
          #some random line
          h = x.dot(w) + b
          #and labels
          label = (h > 0)*1
          y = label
          plt.scatter(x[:, 0], x[:, 1], c=label)
          plt.plot(x[:,0],-w[0]*x[:,0]/w[1]-b/w[1])
          plt.show()
```




```
In [128]: p = Perceptron(w, b, d)
          print("Perceptron of original and predicted values")
          p.fit(x, y, size=1000)
          p.results(x, y)
```

```
Perceptron of original and predicted values
```



4 Independence

For discrete random values:
$E[X * Y] = \sum^{i} \sum^{j} x_i * y_j * f_x y * (x_i, y_j)$
$E[X * Y] = \sum^{i} \sum^{j} x_i * y_j * f_x * (x_i) * f_y * (y_j)$
$E[X * Y] = (\sum^{i} x_i * f_x * (x_i)) * (\sum^{i} y_j * f_y * (y_j))$
$E[X * Y] = E[X]E[Y]$

For continuous random variables:
$E[X * Y] = \int_{\infty}^{-\infty} \int_{\infty}^{\infty} x * y * f_x y * dx * dy$
$E[X * Y] = \int_{\infty}^{-\infty} \int_{\infty}^{\infty} x * y * f_x(x) * f_y(y) * dx * dy$
$E[X * Y] = (\int_{\infty}^{-\infty} x * f_x(x) * dx) * (\int_{\infty}^{\infty} y * f_y(y) * dy)$
E[X*Y] = E[X]E[Y]

5 *Spam filtering equation- (optional bonus problem)

$P(S|W) = \frac{P(S)P(W|S)}{P(W)}$

$P(W) = P(W|H)P(H) + P(W|S)P(S)$

$P(S|W) = \frac{P(S)P(W|S)}{P(W|H)P(H)+P(W|S)P(S)}$

6 I.I.D. assumption in spam filters

1. To disable the spam filter so it lets all spam into the inbox
2. To miss a particular ham email filtered away as spam
3. To get a particular spam into the victim's inbox
4. To get any spam from the victim's inbox