The solution for writen part is also shown within notebook blocks

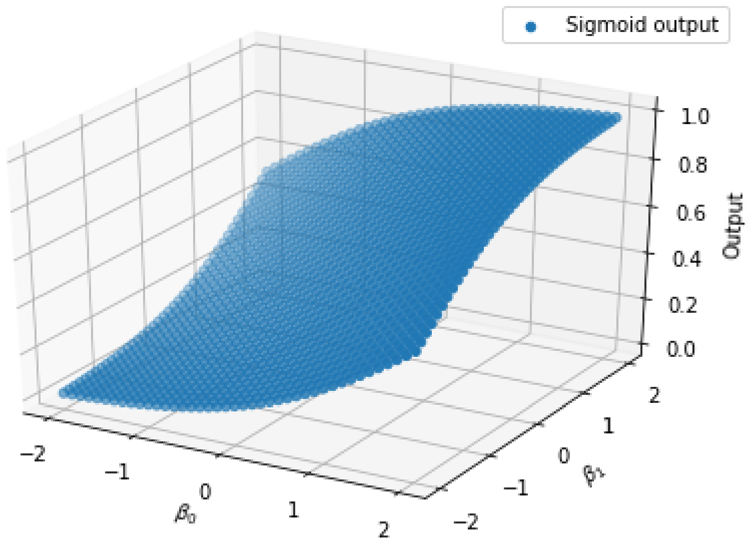
Problem 1

In [18]:
```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
```

In [14]:
```
def sigmoid(x):
  return 1/ (1 + np.exp(-x))
```

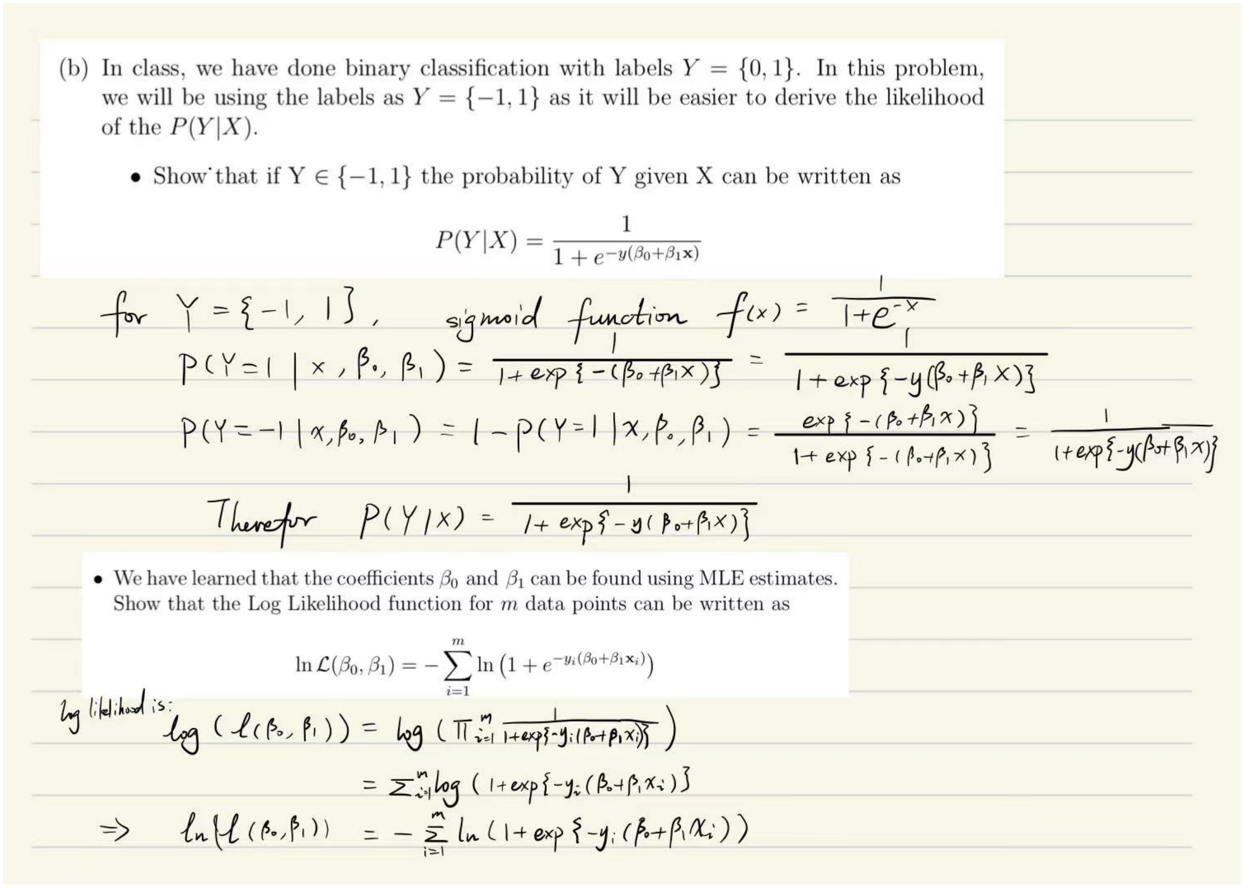
In [15]:
```
beta0s = np.arange(-2,2.1,0.1)
beta1s = np.arange(-2,2.1,0.1)
x = 1
beta0_axis = [beta0s[i] for i in range(len(beta0s)) \
              for j in range(len(beta1s))]
beta1_axis = [beta1s[j] for i in range(len(beta0s)) \
              for j in range(len(beta1s))]
prob = [sigmoid( beta0s[i] + beta1s[j] * x) for i in range(len(beta0s)) \
                for j in range(len(beta1s))]
```

In [26]:
```
fig = plt.figure()
mpl.rcParams['legend.fontsize'] = 10
# ax = fig.add_subplot(111, projection='3d')
ax = fig.gca(projection='3d')
ax.scatter(beta0_axis, beta1_axis, prob, \
           label='Sigmoid output', cmap=cm.Spectral)
ax.set_xlabel('${\\beta_0}$')
ax.set_ylabel('${\\beta_1}$')
ax.set_xticks([-2,-1,0,1,2])
ax.set_yticks([-2,-1,0,1,2])
ax.set_zlabel('Output')
ax.legend()
plt.tight_layout()
plt.show()
```



In [ ]:
```
from IPython.display import Image
Image(filename='problem1b.jpg')
```

Out[55]:



(b) In class, we have done binary classification with labels $Y = \{0, 1\}$. In this problem, we will be using the labels as $Y = \{-1, 1\}$ as it will be easier to derive the likelihood of the $P(Y|X)$.

- Show that if Y $\in \{-1, 1\}$ the probability of Y given X can be written as

$$P(Y|X) = \frac{1}{1 + e^{-y(\beta_0 + \beta_1 \mathbf{x})}}$$

for $Y = \{-1, 1\}$, sigmoid function $f(x) = \frac{1}{1+e^{-x}}$

$$P(Y=1 \mid x, \beta_0, \beta_1) = \frac{1}{1+\exp\{-(\beta_0+\beta_1 X)\}} = \frac{1}{1+\exp\{-y(\beta_0+\beta_1 X)\}}$$

$$P(Y=-1 \mid x, \beta_0, \beta_1) = 1 - P(Y=1 \mid x, \beta_0, \beta_1) = \frac{\exp\{-(\beta_0+\beta_1 x)\}}{1+\exp\{-(\beta_0+\beta_1 x)\}} = \frac{1}{1+\exp\{-y(\beta_0+\beta_1 x)\}}$$

Therefor $P(Y|X) = \frac{1}{1+\exp\{-y(\beta_0+\beta_1 X)\}}$

- We have learned that the coefficients β_0 and β_1 can be found using MLE estimates. Show that the Log Likelihood function for m data points can be written as

$$\ln \mathcal{L}(\beta_0, \beta_1) = -\sum_{i=1}^{m} \ln\left(1 + e^{-y_i(\beta_0 + \beta_1 \mathbf{x}_i)}\right)$$

log likelihood is:

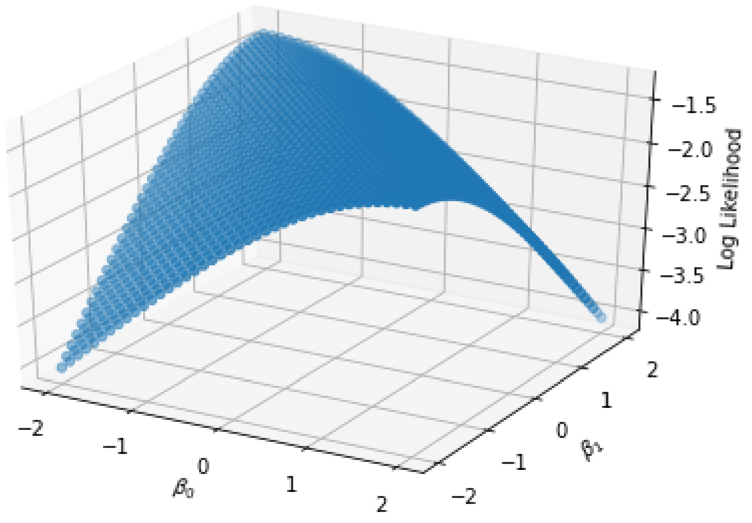
$$\log(\ell(\beta_0, \beta_1)) = \log\left(\prod_{i=1}^{m} \frac{1}{1+\exp\{-y_i(\beta_0+\beta_1 x_i)\}}\right)$$

$$= \sum_{i=1}^{m} \log(1+\exp\{-y_i(\beta_0+\beta_1 x_i)\}$$

$$\Rightarrow \quad \ln\{\ell(\beta_0, \beta_1)) = -\sum_{i=1}^{m} \ln(1+\exp\{-y_i(\beta_0+\beta_1 x_i))$$

In [8]:
```
def log_likelihood(beta0,beta1):
  # return - np.log( 1 + np.exp( -y * (beta0 + beta1 * x) ) )
  return -np.log(1 + (np.exp(beta0 + beta1))) +\
        -np.log(1 + (np.exp(-(beta0 + beta1))))
```

In [9]:
```
fig = plt.figure()
# x, y = 1, -1
beta0_axis = [beta0s[i] for i in range(len(beta0s)) \
              for j in range(len(beta1s))]
beta1_axis = [beta1s[j] for i in range(len(beta0s)) \
              for j in range(len(beta1s))]
ll_func = [log_likelihood(beta0s[i],beta1s[j]) \
           for i in range(len(beta0s)) for j in range(len(beta1s))]
ax1 = fig.gca(projection='3d')
ax1.scatter(beta0_axis, beta1_axis, ll_func)
ax1.set_xlabel('${\\beta_0}$')
ax1.set_ylabel('${\\beta_1}$')
ax1.set_xticks([-2,-1,0,1,2])
ax1.set_yticks([-2,-1,0,1,2])
ax1.set_zlabel('Log Likelihood')
# ax1.legend()
plt.tight_layout()
plt.show()
```



Based on the graph, it is possible to maximize the Log Likelihood funciton because it has an optima.

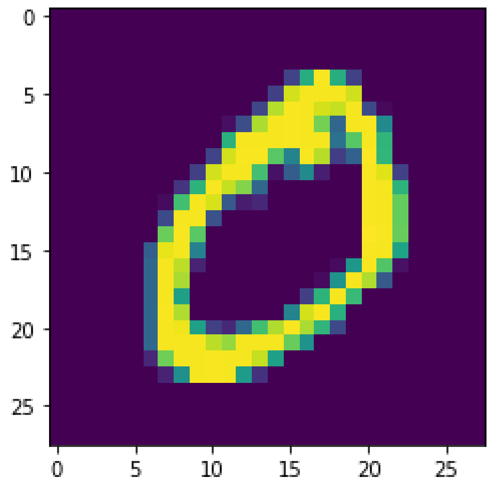
Problem 2

In [ ]:
```
data = np.load( "data.npy" )
label = np.load("label.npy")
data.shape # (14780,784)
label.shape
```

Out[41]:
```
(14780,)
```

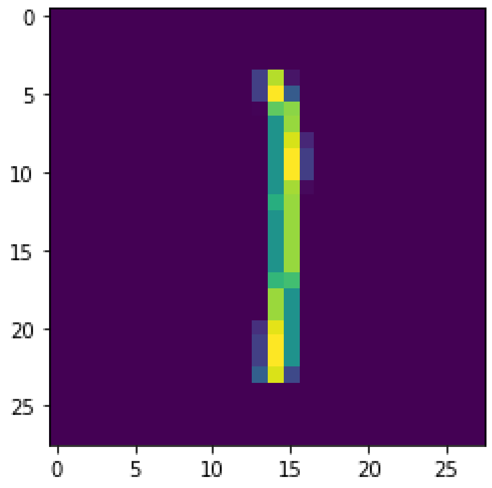
In [ ]:
```
zero_idx = np.where(label==0)[0][0]
one_idx = np.where(label==1)[0][0]
plt.imshow(data[zero_idx,:].reshape(28, 28))
```

Out[48]:
```
<matplotlib.image.AxesImage at 0x7f2a448d70d0>
```



In [ ]:
```
plt.imshow(data[one_idx,:].reshape(28, 28))
```

Out[49]:
```
<matplotlib.image.AxesImage at 0x7f2a42e13190>
```



In [ ]:
```
def normalize(a):
  minval = min(a.flatten())
  maxval=max(a.flatten())
  a = (a-minval)/(maxval-minval)
  return a
data = normalize(data)
```

In [ ]:
```
# change to -1 (label 1) & 1(label 0)
zero_idx = np.where(label==0)[0]
one_idx = np.where(label==1)[0]
label[zero_idx] = 1
label[one_idx] = -1
```

Why random splitting better than sequential splitting in our case

Because the label 0 and label 1 are not randomly distributed, images with label 1 are in the end of the data array, so for a sequential splitting, images in the test set would be in the same category.

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data, label,
                                                    test_size=0.2, random_state
```

In [ ]:
```
mu, sigma = 0, 1
beta0_ = np.random.normal(mu, sigma)
d = 784
beta_ = np.random.normal(mu, sigma, d)
```

Function to compute log loss

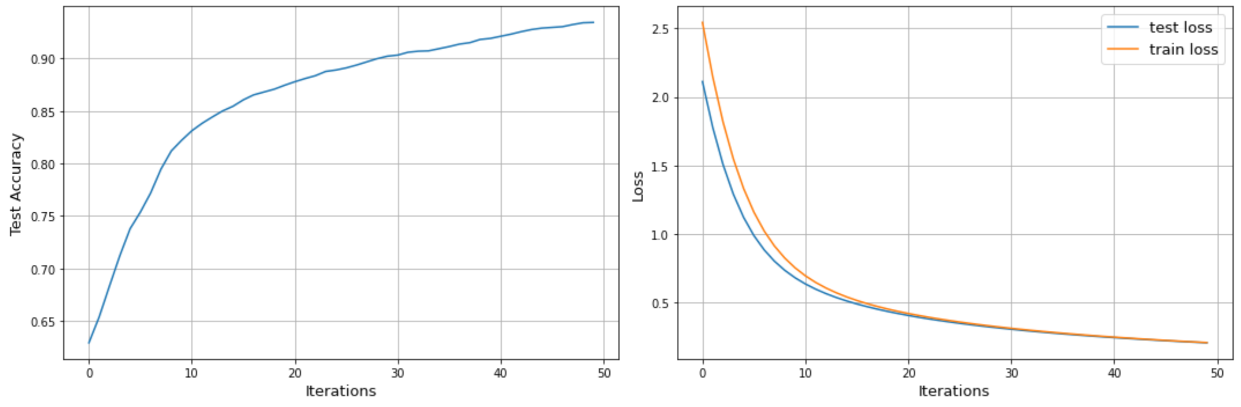
In [ ]:
```
def compute_loss(data, labels, beta_, beta0_):
  m = data.shape[0]
  total = 0
  for i in range(m):
    total += np.log(1 + np.exp(-labels[i] * (beta0_ + beta_@data[i])))
  return 1/m * total
def compute_gradient(data, labels, beta_, beta0_):
  m = data.shape[0]
  totalb, totalb0 = 0, 0
  for i in range(m):
    numer = np.exp(-labels[i] * (beta0_ + beta_@data[i]))
    denom = np.exp(- (labels[i] * (beta0_ + beta_@data[i])))
    totalb0 += (numer / (1 + denom)) * labels[i]
    totalb += (numer / (1 + denom)) * labels[i] * data[i]
  dB = -1/m * totalb
  dB_0 = -1/m * totalb0
  return dB, dB_0

def predict(X_test, y_test, beta_, beta0_):
  tmp = beta0_ + beta_ @ X_test.T
  sigmoid_output = 1/(1 + np.exp(-tmp))
  # print(sigmoid_output.shape)
  pred = [1 if p>=0.5 else -1 for p in sigmoid_output]
  y_pred = np.array(pred)
  cnt = 0
  for i in range(len(y_pred)):
    if y_test[i] == y_pred[i]:
      cnt+=1
  return cnt/len(y_pred)
```

In [ ]:
```
lr = 0.05
test_acc, test_loss, train_loss = [], [], []
for _ in range(50):
  loss = compute_loss(X_train, y_train, beta_, beta0_)
  train_loss.append(loss)
  # print(loss)
  dB, dB_0 = compute_gradient(X_train, y_train, beta_, beta0_)
  # print(dB.shape, dB_0.shape)
  beta_ -= lr * dB
  beta0_ -= lr * dB_0

  accuracy_test = predict(X_test, y_test, beta_, beta0_)
  loss_test = compute_loss(X_test, y_test, beta_, beta0_)
  test_acc.append(accuracy_test)
  test_loss.append(loss_test)
```

In [ ]:
```
fig, ((ax1, ax2))= plt.subplots(1,2,figsize = (15,5))
ax1.grid()
ax2.grid()
plt.rcParams['font.family'] = "sans-serif"
plt.rcParams['font.sans-serif'] = ['Times New Roman']
x_axis = [_ for _ in range(50)]
# print(len(test_acc))
ax1.plot(x_axis, test_acc)
ax1.set_xlabel('Iterations',fontsize = 13)
ax1.set_ylabel('Test Accuracy',fontsize = 13)
ax2.plot(x_axis, test_loss, label = 'test loss')
ax2.plot(x_axis, train_loss, label = 'train loss')
ax2.set_xlabel('Iterations',fontsize = 13)
ax2.set_ylabel('Loss',fontsize = 13)
ax2.legend(fontsize = 13)
plt.tight_layout()
plt.show()
```



classification rule for the threshold 0.5

$Y = 1$ when $P(Y = 1|X) \geq 0.5$

Since P(Y|X) = $\frac{1}{1+e^{-y(\beta_0+\beta_1 x)}}$

$Y = 1$ when $P\frac{1}{1+e^{-(\beta_0+\beta_1 x)}} \geq 0.5$

Equals to $e^{-(\beta_0+\beta_1 x)} \leq 1$

Equals to $(\beta_0 + \beta_1 x) \geq 0$

Thus $Y = 1$ when $(\beta_0 + \beta_1 x) \geq 0$

Problem 3

1. **Design P (Y = y|X = x) such that (i) P (y = 0) = P (y = 1) = 0.5; and (ii) the classification accuracy of any classifier is at most 0.9; and (iii) the accuracy of the Bayes optimal possible classifier is at least 0.8.**

My distribution is defined as follows:

When x > 0.5: P(Y=1|X) = 0.1, P(Y=0|X) = 0.9

When x <= 0.5: P(Y=1|X) = 0.9, P(Y=0|X) = 0.1

In [ ]:
```
from sklearn.linear_model import LogisticRegression
# from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
```

In [ ]:
```
def gen_y(p):
  tmp = np.random.random()
  if tmp > p:
    return 1
  return 0
```

In [ ]:
```
def gen_distribution(length):
    x, y = [], []
    for i in range(length):
        dice = np.random.uniform(0,1)
        x.append([dice])
        if dice > 0.5:
          y.append(gen_y(0.9))
        else:
          y.append(gen_y(0.1))
    return x, y
```

2. Using Python, generate n = 100 training data points according to the distribution you designed above and train a binary classifier using logistic regression on training data.

In [ ]:
```
x_100,y_100 = gen_distribution(100)
clf = LogisticRegression()
clf.fit(x_100, y_100)
```

Out[42]:
```
LogisticRegression()
```

3. Generate and n = 100 test data points according to the distribution you designed in part 1 and compute the prediction accuracy (on the test data) of the classifier that you designed in part 2.

Also, compute the accuracy of the Bayes optimal classifier on the test data. Why do you think Bayes optimal classifier is performing better?

In [ ]:
```
x_test, y_test = gen_distribution(100)
y_pred = clf.predict(x_test)
print('prediction accuracy (on the test data) of the Logistic classifier',\
      accuracy_score(y_pred, y_test))
```

```
prediction accuracy (on the test data) of the Logistic classifier 0.8
```

In [ ]:
```
def bayes_clf(x):
    y_pred = [0 if cur[0] > 0.5 else 1 for cur in x]
    return y_pred
```

In [ ]:
```
y_pred = bayes_clf(x_test)
print('prediction accuracy (on the test data) of the Bayes classifier',\
      accuracy_score(y_pred, y_test))
```

```
prediction accuracy (on the test data) of the Bayes classifier 0.88
```

Why do you think Bayes optimal classifier is performing better?

Because the bayes optimal classifier is based on a rule, which is designed according to the true distribution of data.

4. Redo parts 2,3 with n = 1000. Are the results any different than part 3? Why?

In [ ]:
```
x_1000,y_1000 = gen_distribution(1000)
clf = LogisticRegression()
clf.fit(x_1000, y_1000)
x_test, y_test = gen_distribution(1000)
y_pred = clf.predict(x_test)
print('prediction accuracy (on the test data) of the Logistic classifier',\
      accuracy_score(y_pred, y_test))
y_pred = bayes_clf(x_test)
print('prediction accuracy (on the test data) of the Bayes classifier',\
      accuracy_score(y_pred, y_test))
```

```
prediction accuracy (on the test data) of the Logistic classifier 0.846
prediction accuracy (on the test data) of the Bayes classifier 0.88
```

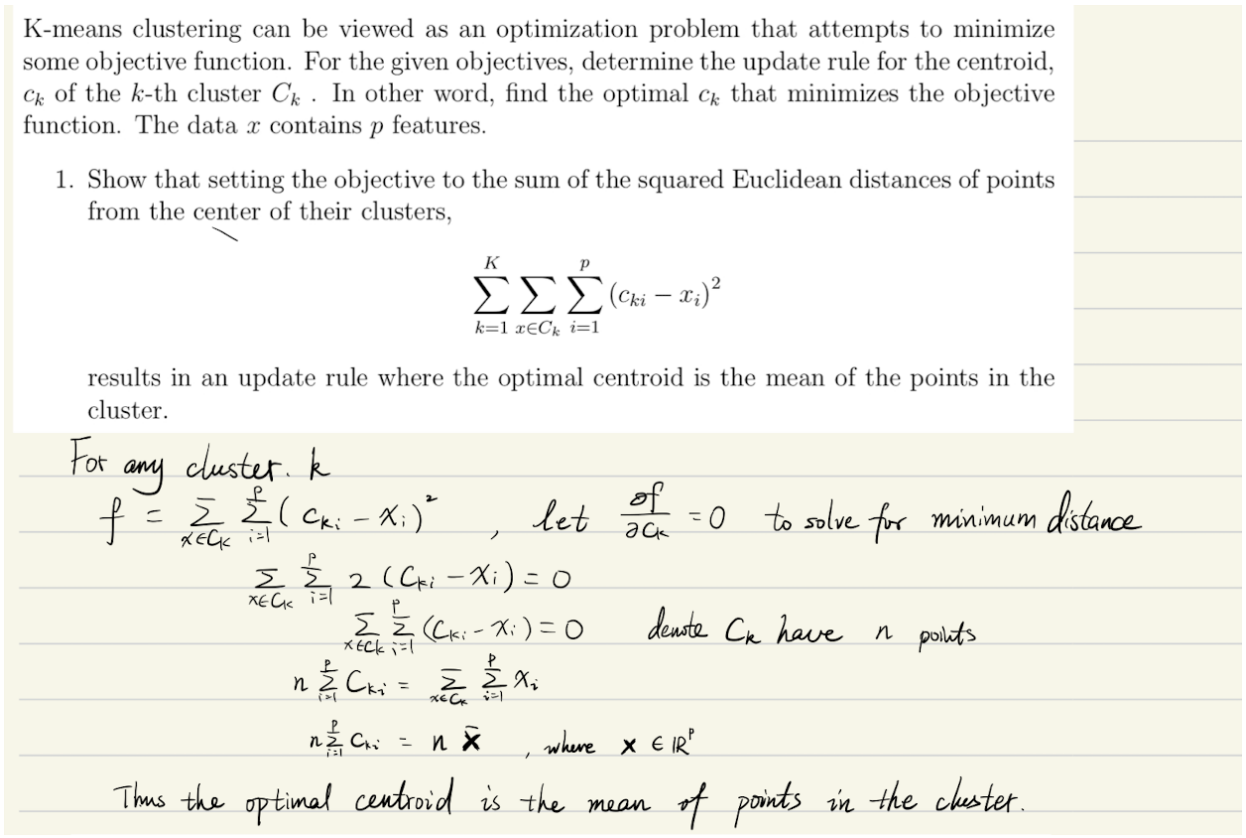
Are the results any different than part 3? Why?

The accuracy of the Bayes classifier is still higher. But the accuracy of the logistic regression classifier inceases a little. Because with more data points, the logistic regressor learns more about the true distribution. The accuracy of the Bayes classifier did not change because more data point does not change the rule or the true distribution.

Problem 4

In [11]:
```
from IPython.display import Image
Image(filename='hw5-41.png')
```

Out[11]:



K-means clustering can be viewed as an optimization problem that attempts to minimize some objective function. For the given objectives, determine the update rule for the centroid, c_k of the k-th cluster C_k . In other word, find the optimal c_k that minimizes the objective function. The data x contains p features.

1. Show that setting the objective to the sum of the squared Euclidean distances of points from the center of their clusters,

$$\sum_{k=1}^{K}\sum_{x\in C_k}\sum_{i=1}^{p}(c_{ki}-x_i)^2$$

results in an update rule where the optimal centroid is the mean of the points in the cluster.

For any cluster. k

$$f = \sum_{x\in C_k}\sum_{i=1}^{p}(c_{ki}-x_i)^2 \quad , \quad \text{let } \frac{\partial f}{\partial c_k} = 0 \text{ to solve for minimum distance}$$

$$\sum_{x\in C_k}\sum_{i=1}^{p} 2(c_{ki}-x_i) = 0$$

$$\sum_{x\in C_k}\sum_{i=1}^{p}(c_{ki}-x_i) = 0 \qquad \text{denote } C_k \text{ have } n \text{ points}$$

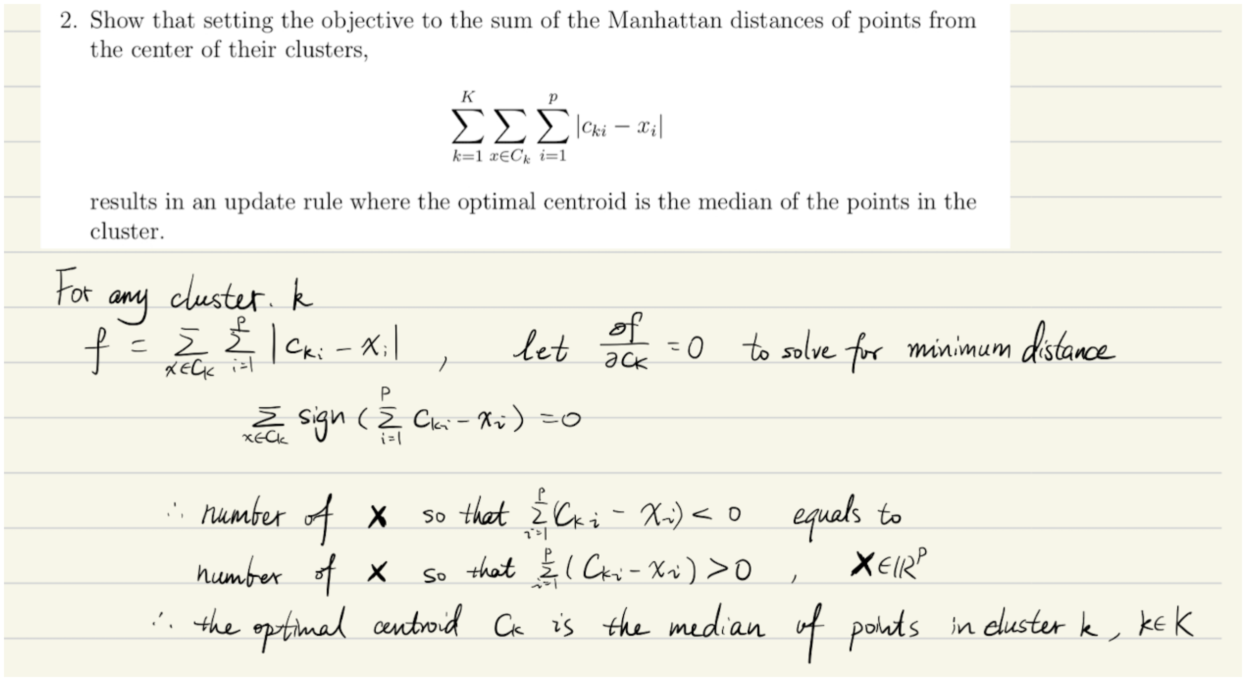
$$n\sum_{i=1}^{p} c_{ki} = \sum_{x\in C_k}\sum_{i=1}^{p} x_i$$

$$n\sum_{i=1}^{p} c_{ki} = n\bar{x} \quad , \text{ where } x \in \mathbb{R}^p$$

Thus the optimal centroid is the mean of points in the cluster.

In [12]:
```
Image(filename='hw5-42.png')
```

Out[12]:



2. Show that setting the objective to the sum of the Manhattan distances of points from the center of their clusters,

$$\sum_{k=1}^{K}\sum_{x\in C_k}\sum_{i=1}^{p}|c_{ki}-x_i|$$

results in an update rule where the optimal centroid is the median of the points in the cluster.

For any cluster. k

$$f = \sum_{x\in C_k}\sum_{i=1}^{p}|c_{ki}-x_i| \quad , \quad \text{let } \frac{\partial f}{\partial c_k} = 0 \text{ to solve for minimum distance}$$

$$\sum_{x\in C_k} \text{sign}\left(\sum_{i=1}^{p} c_{ki}-x_i\right) = 0$$

∴ number of x so that $\sum_{i=1}^{p}(c_{ki}-x_i) < 0$ equals to

number of x so that $\sum_{i=1}^{p}(c_{ki}-x_i) > 0$, $x \in \mathbb{R}^p$

∴ the optimal centroid c_k is the median of points in cluster k, $k \in K$