Problem Set 2 Part 2

By: Kali Benavides

Collaborators: Lan Ha, Les Armstrong, Serena Patel, Alexa Canaan, Allison Bell

Problem 2.3

A)

$$P(Y=1 \mid X=x) = p(x)$$

$$P(Y=0 \mid X=x) = (1-p(x))$$

$$f(y) = p(x)^{y}(1-p(x))^{1-y}$$

For all x_i & corresponding y_i

$$[p(x_1)^{y_1}(1-p(x_1))^{1-y_1}][p(x_2)^{y_2}(1-p(x_2))^{1-y_2}]\ldots[p(x_n)^{y_n}(1-p(x_n))^{1-y_n}]$$

$$\prod_{i=1}^{n} p(x_i)^{y_i}(1-p(x_i))^{1-y_i} \rightarrow \text{Likelihood function}$$

$$\prod_{i=1}^{n} \log\left(p(x_i)^{y_i}(1-p(x_i))^{1-y_i}\right) \rightarrow \text{Log likelihood function}$$

$$\sum_{i=1}^{n}\left[y_i \log(p(x_i)) + (1-y_i)\log(1-p(x_i))\right]$$

$$\sum_{i=1}^{n}\left[y_i \log(p(x_i)) + \log(1-p(x_i)) - y_i \log(1-p(x_i))\right]$$

$$\sum_{i=1}^{n}\log(1-p(x_i)) + \sum_{i=1}^{n} y_i \log(p(x_i)) - y_i \log(1-p(x_i))$$

$$\sum_{i=1}^{n}\log(1-p(x_i)) + \sum_{i=1}^{n} y_i \log\left(\frac{p(x_i)}{1-p(x_i)}\right)$$

Logistic Model

$$\log\left(\frac{p(x_i)}{1-p(x_i)}\right) = \beta_0 + x_i\beta$$

$$\text{if } p(x_i) = \frac{1}{1+e^{-(\beta_0 + x\beta)}}$$

$$= 1+e^{-(\beta_0 + x\beta)}.$$

$$\sum_{i=1}^{n} \log\left(1-\left(\frac{1}{1+e^{-(\beta_0+x_i\beta)}}\right)\right) + \sum_{i=1}^{n} y_i(\beta_0+x_i\beta)$$

$$\log\left(\frac{1+e^{-(\beta_0+x_i\beta)}-1}{1+e^{-(\beta_0+x_i\beta)}}\right)$$

$$\log\left(\frac{e^{-(\beta_0+x_i\beta)}}{1+e^{-(\beta_0+x_i\beta)}}\right)$$

$$\log\left(e^{-(\beta_0+x_i\beta)}\right) - \log\left(1+e^{-(\beta_0+x_i\beta)}\right)$$

$$-(\beta_0+x_i\beta) - \log\left(1+e^{-(\beta_0+x_i\beta)}\right)$$

$$\sum_{i=1}^{n} \left(-\log\left(1+e^{(\beta_0+x_i\beta)}\right) + y_i(\beta_0+x_i\beta)\right)$$

B) $$P(Y=c \mid X=x) = \frac{P(Y=c)P(X=x \mid Y=c)}{P(X=x)} \sim P(Y=c)P(X=x \mid Y=c)$$

$c = 0$ or 1

$P(Y=1) = \eta \quad P(Y=0) = 1-\eta$

$P(X=x \mid Y=c) \sim N(\mu_c, \Sigma)$

Gaussian distribution:

$$f_x(x) = \frac{1}{\sqrt{2\pi}} \frac{1}{\sqrt{\det(\Sigma)}} \exp\left(-(x-\mu_c)^T \Sigma^{-1} (x-\mu_c)\right)$$

$$P(Y=1 \mid X=x) \propto \eta \left[\frac{1}{\sqrt{2\pi}} \frac{1}{\sqrt{\det(\Sigma)}} \exp\left(-(x-\mu_c)^T \Sigma^{-1} (x-\mu_c)\right)\right]$$

$$P(Y=0 \mid X=x) \propto (1-\eta) \left[\frac{1}{\sqrt{2\pi}} \frac{1}{\sqrt{\det(\Sigma)}} \exp\left((x-\mu_c)^T \Sigma^{-1} (x-\mu_c)\right)\right]$$

For all x_i, y_i outcomes...

$$\underbrace{\prod_{i=1}^{n} \left(\frac{\eta}{\sqrt{2\pi}\sqrt{\det\Sigma}} \exp\left(-(x_i-\mu_c)^T \Sigma^{-1} (x_i-\mu_c)\right)\right)\left(\frac{(1-\eta)}{\sqrt{2\pi}\sqrt{\det\Sigma}} \exp\left(-(x-\mu_c)^T \Sigma^{-1} (x-\mu_c)\right)\right)}_{\text{Likelihood function}}$$

$$\sum_{i=1}^{n} \Big[\log\left(\frac{\eta}{\sqrt{2\pi}\sqrt{\det\Sigma}}\right) - (x_i-\mu_c)^T \Sigma^{-1} (x_i-\mu_c) \Big] \Big\rangle \text{Log likelihood function}$$

$$+ \underbrace{\log\left(\frac{1-\eta}{\sqrt{2\pi}\sqrt{\det\Sigma}}\right)}_{\text{Constant terms}} - (x_i-\mu_c)^T \Sigma^{-1} (x_i-\mu_c) \Big]$$

→ We will drop these terms because we are looking at the proportionality of $P(Y \mid X)$

$$\sum_{i=1}^{n}\Big[\Big(x_i^T\Sigma^{-1}x_i - \mu^T\Sigma^{-1}x_i - x_i^T\Sigma^{-1}\mu + \mu^T\Sigma^{-1}\mu$$

$$- x_i^T\Sigma^{-1}x_i + \mu^T\Sigma^{-1}x_i - x_i^T\Sigma^{-1}\mu + \mu^T\Sigma^{-1}\mu\Big]$$

$$P(y|x) \propto \sum_{i=1}^{n} -\mu^T\Sigma^{-1}x_i - x_i^T\Sigma^{-1}\mu + \mu^T\Sigma^{-1}\mu$$

$$+ \mu^T\Sigma^{-1}x_i - x_i^T\Sigma^{-1}\mu + \mu^T\Sigma^{-1}\mu$$

C) The decision boundary for Logistic regression is linear.

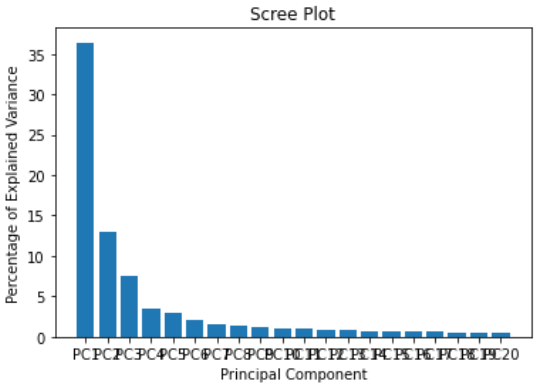
It relates to when $\beta_o + x\beta > 0\ i.e. Y = 1\ because\ the\ Probability\ \geq 0.5$

when $\beta_o + x\beta < 0\ i.e. Y = 0\ because\ the\ Probability\ < 0.5$

For Linear discriminant analysis it also has the same linear decision boundary. It becomes a linear expression when we make the assumption that the covariance matrix is the same for all classes.

Problem 2.4

A) To determine the number of cell clusters I first used PCA to visualize the clustering of the 272 cells, using both PC1, PC2 and PC3. These three PCs explained approximately 56% of the variance. 20 Principle Components explained 77% of the variance as plotted below in a Scree Plot. I then graphed PC1 and PC2 to visualize in 2 dimensions. And then graphed PC1, PC2 and PC3 to visualize in 3 dimensions. This did not provide very helpful clustering information so I performed tSNE and graphed the visualization using tSNE to reduce the data to two dimensions. The tSNE graph indicated there were about 3-5 potential clusters. I then ran K Means for a range of cluster numbers from 2 to 9. Viewing the plot of the inertia plotted as a function of the number of clusters, indicated that the elbow occurred at 5 clusters. The calculation of the silhouette score for each number of clusters also indicated that 5 clusters had the value closest to 1. (In this case still only 0.137). This resulted in a KMeans clustering with an inertia (Within cluster sum of squares) of 0.048.




```
Explained variation from 20 principal component: 0.771493844098977
```

Below: Loading in data and transposing the data so that the samples are the rows of the array and the parameters are as columns so that I can utilize the PCA and tSNE built in python functions.

```
time_start = time.time()

df = pd.read_csv(r'C:\Users\kalib\Documents\IDS.131\Pset 2\trapnell.csv', sep=',')

print('Data loading into Dataframe is done Time elapsed {}'.format(time.time()-time_start))
```

```
Data loading into Dataframe is done Time elapsed 71.92458033561707
```

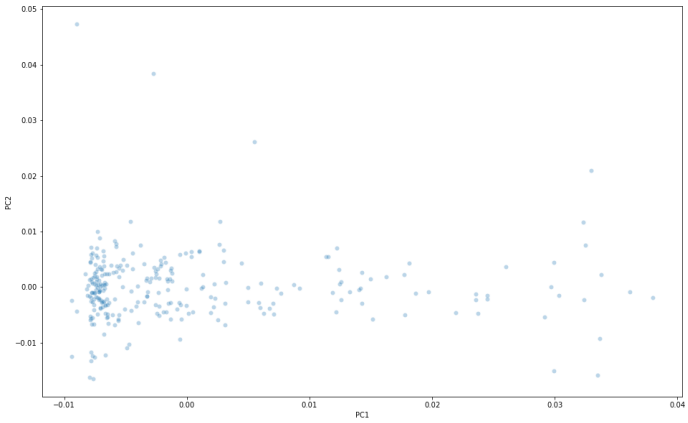
```
df.shape
```

```
(1065023, 272)
```

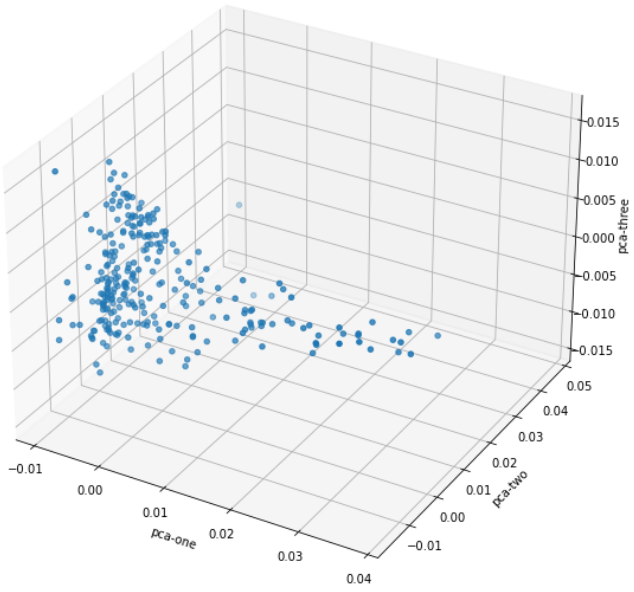
```
#Expects samples to be rows,

time_start = time.time()
df_transpose = df.T
print(df_transpose.shape)
print('Transposing data is done Time elapsed {}'.format(time.time()-time_start))
```

```
(272, 1065023)
Transposing data is done Time elapsed 0.046120643615722656
```

Graph of Principle Component 1 versus Principle Component 2



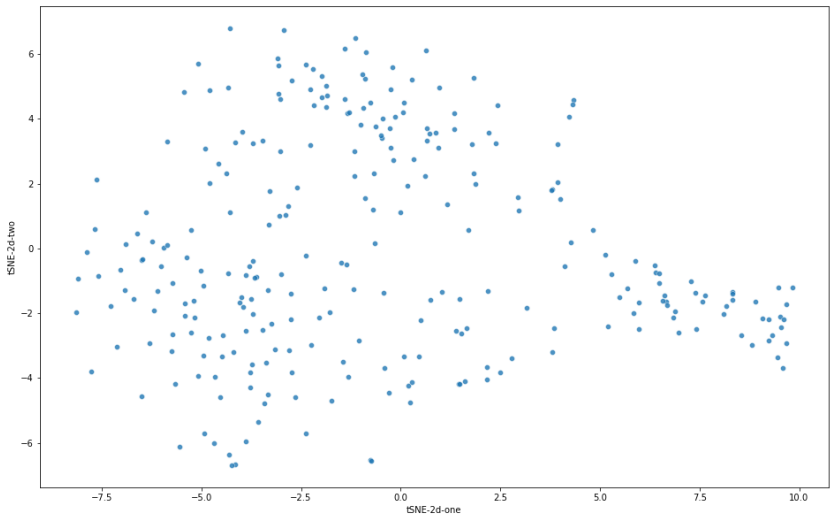
Graph of Principle Component 1, Principle Component 2 and Principle Component 3



After performing tSNE

```
#tSNE on the data to visualize the clusters

tsne = TSNE(n_components=2, perplexity=50, n_iter=500, learning_rate = 100, init='random')
tsne_results = tsne.fit_transform(df_transpose)

tsne_results.shape
```

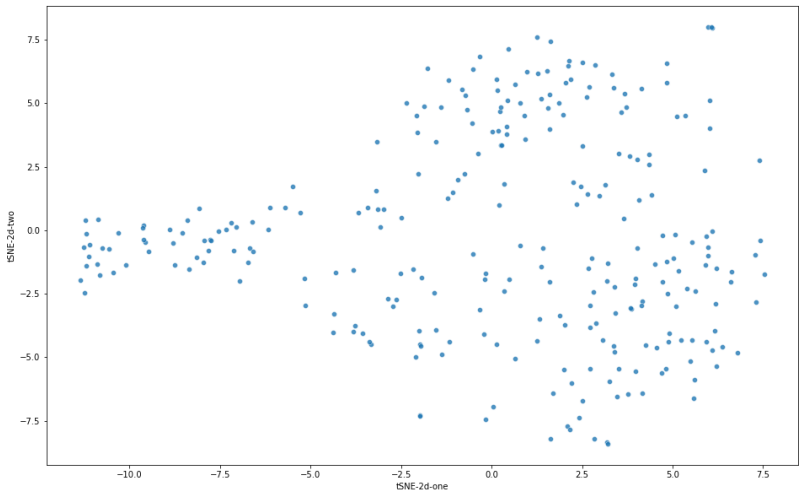
```
(272, 2)
```



tSNE after performing PCA to 20 Principle Components

```
#tSNE on the data to visualize the clusters

tsne = TSNE(n_components=2, perplexity=50, n_iter=500, learning_rate = 200, init='random')
tsne_results = tsne.fit_transform(principalComponents_Trapnell)

tsne_results.shape
```

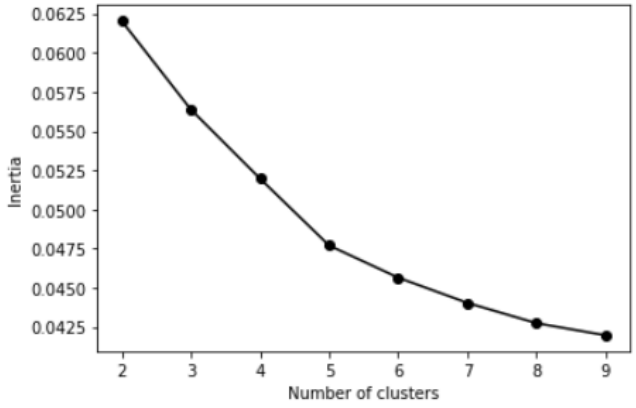
```
(272, 2)
```



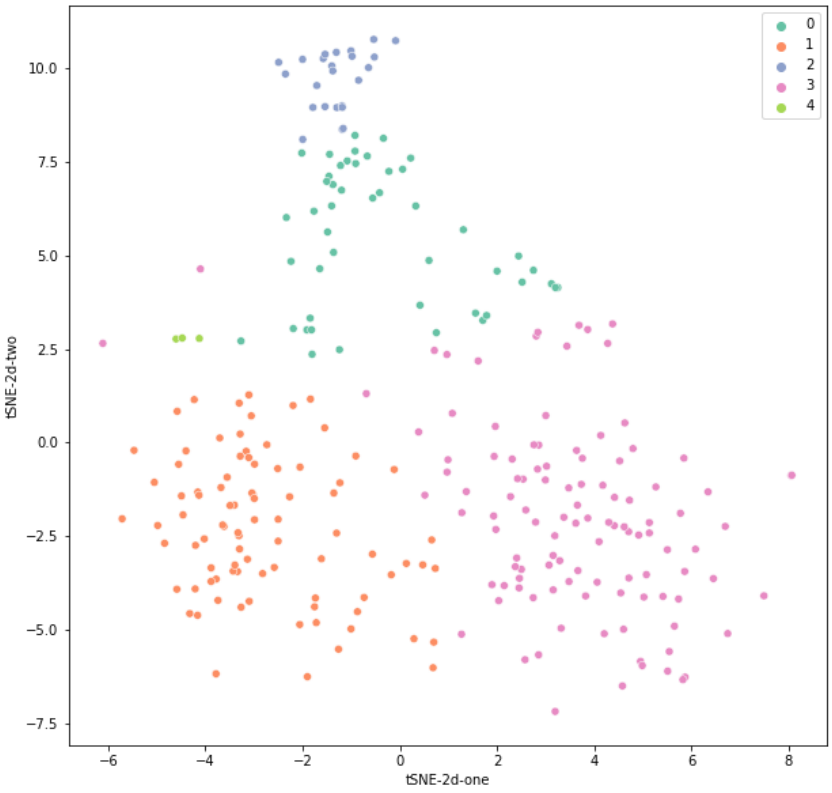
Calculating the Kmeans cluster number to select

```
inertias = []
silhouette_scores = []

ks = range(2,15)
data = df_transpose

for k in ks:
    model = KMeans(n_clusters=k)

    model.fit(data)
    labels = model.labels_

    inertias.append(model.inertia_)
    ss = metrics.silhouette_score(data, labels)   # calculate silhouette_score
    silhouette_scores.append(ss)      # store all the scores
    print('Parameter:', k, 'Score', ss)

# converting the results into a dataframe and plotting them
plt.plot(ks, inertias, '-o', color='black')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
```

```
Parameter: 2 Score 0.36263228584296775
Parameter: 3 Score 0.1364409480952796
Parameter: 4 Score 0.13580219052705544
Parameter: 5 Score 0.1369690204951735
Parameter: 6 Score 0.11541305441161076
Parameter: 7 Score 0.11484078707179597
Parameter: 8 Score 0.10450342926944849
Parameter: 9 Score 0.08120395397319648

Text(0, 0.5, 'Inertia')
```



K-Means Clustering with selection of 5 clusters




```
k_means = KMeans(init="k-means++", n_clusters = 5, n_init=12 )
```

```
y = k_means.fit(df_transpose)
```

```
k_means_cluster_centers = y.cluster_centers_
print(k_means_cluster_centers)
```

```
[[0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 5.16987883e-25
  5.16987883e-25 5.16987883e-25]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 6.20385459e-25
  6.20385459e-25 6.20385459e-25]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 3.10192730e-25
  3.10192730e-25 3.10192730e-25]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 1.03397577e-25
  1.03397577e-25 1.03397577e-25]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 1.54076521e-09
  1.54076521e-09 1.54076521e-09]]
```

```
labels = y.labels_
print(labels)
```

```
[4 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 4 1 1 1 1 1 4
 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 4 1 1 4
 2 0 0 0 0 4 4 4 2 4 2 0 2 4 4 1 4 4 1 0 0 4 4 4 4 1 0 0 0 1 1 4 1 4 0 0 4 2
 4 1 2 4 4 1 0 4 0 4 0 4 4 4 4 1 2 0 1 4 0 0 4 0 1 1 2 4 4 0 0 0 4 4 2 4 4
 2 1 4 3 4 4 0 4 4 2 2 4 0 4 4 4 4 2 4 2 4 4 0 4 0 4 4 4 0 4 4 4 0 4 4 4 2
 4 4 4 4 4 4 2 0 4 0 0 4 4 0 3 4 0 4 0 0 0 4 4 4 4 4 2 4 4 4 4 2 4 4 4 4 0 4
 4 4 0 4 4 4 0 0 0 4 0 1 2 4 4 4 4 4 4 1 4 0 4 0 4 4 2 1 4 4 4 4 0 4 4 2 0
 1 4 4 4 4 4 1 1 1 0 4 4 4]
```

```
k_means_inertia = k_means.inertia_
print(k_means_inertia)
```

```
0.047696164711497284
```

B) To train a classifier on the data I first utilized the labels from the KMeans clustering to assign each of the 272 cells to a cluster either 0,1,2,3 or 4 (5 total clusters).
I then ran logistic regression one at a time to determine the prediction of one class being the classifier versus any other class. I set this up by adding additional columns for each cluster number which were set up with options of binary class designations (0 or 1). I then split the data into a training set and testing set (80/20 split). I used Ridge regression as the regularization to control for high coefficient values and to minimize the influence of parameters which were not highly correlated with the classification since there were over a million different gene expressions as parameters. For each class I ran logistic regression with the corresponding column where 1 was assigned if the sample was assigned to that class and zero if it was assigned to another class. This resulted in a model for each cluster with coefficients corresponding to each parameter, in this case the gene expression data. I then selected the gene expression corresponding to the largest coefficient to serve as the marker gene. The largest coefficient parameter indicates the parameter which has a large effect on the classification for that cell. For two of the clusters they shared the same two largest coefficients so I selected the third largest coefficient to serve as a unique gene marker.
The accuracy test score as shown below ranged from 0.9 to 1.0.

Cluster No.	**Marker Gene**	**Explanation**
0	Col.# 106178	The first two largest coefficients were the same as the first two largest coefficients for cluster 2 so I selected the third largest coefficient as the marker gene for this cluster.
1	Col# 94838	I selected the gene expression parameter associated with the largest coefficient for this logistic regression model
2	Col. # 173840	The first two largest coefficients were the same as the first two largest coefficients for cluster 0 so I selected the third largest coefficient as the marker gene for this cluster.
3	Col # 667	I selected the gene expression parameter associated with the largest coefficient for this logistic regression model
4	Col# 204511	I selected the gene expression parameter associated with the largest coefficient for this logistic regression model

```
df_transpose['Label'] = labels

df_transpose['0'] = np.where(df_transpose['Label']==0, 1, 0)
df_transpose['1'] = np.where(df_transpose['Label']==1, 1, 0)
df_transpose['2'] = np.where(df_transpose['Label']==2, 1, 0)
df_transpose['3'] = np.where(df_transpose['Label']==3, 1, 0)
df_transpose['4'] = np.where(df_transpose['Label']==4, 1, 0)



df_transpose.shape
df_transpose.head()
```

	0	1	2	3	4	5	6	7	8	9	...	1065019	1065020	1065021	1065022	Label	0	1	2	3	4
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	4	0	0	0	0	1
0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1	0	1	0	0	0
0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1	0	1	0	0	0
0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1	0	1	0	0	0
0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1	0	1	0	0	0

```
#Function to run logistic regression on each cluster testing it to see if it is more likely
#to belong to the cluster it was labeled as or some other cluster
def LogRegCluster(Class):
    time_start = time.time()
    y = df_transpose[Class]
    X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=4)
    print ('Train set:', X_train.shape,  y_train.shape)
    print ('Test set:', X_test.shape,  y_test.shape)
    #Instantiate model
    logreg = LogisticRegression(penalty ='l2', solver = 'liblinear', class_weight = 'balanced')
    #Fit the model with the data
    logreg.fit(X_train, y_train)
    #Print out predicted y values
    y_pred_train = logreg.predict(X_train)
    y_pred_test = logreg.predict(X_test)
    print('Shape of prediction for training set', y_pred_train.shape)
    print('Shape of prediction for test set',y_pred_test.shape)
    Coeff = logreg.coef_
    max_index = np.argmax(Coeff)
    print('Column of maximum coefficient value is', max_index)
    print('Maximum coefficient value is', Coeff[0,max_index])
    idx = (-Coeff).argsort()[:5]
    print('Sorted Coeff values from largest to smallest', idx)
    ascore = accuracy_score(y_test, y_pred_test)
    print('Accuracy Test Score is ', ascore)
    print('Logistic Regression is done for Cluster Time elapsed {}'.format(time.time()-time_start))
    return Coeff
```

```
Cluster0 = LogRegCluster('0')
```

```
Train set: (217, 1065023) (217,)
Test set: (55, 1065023) (55,)
Shape of prediction for training set (217,)
Shape of prediction for test set (55,)
Column of maximum coefficient value is 13894
Maximum coefficient value is 0.45461571171377624
Sorted Coeff values from largest to smallest [[ 13894 304794 106178 ... 152051 204511  94838]]
Accuracy Test Score is  0.9090909090909091
Logistic Regression is done for Cluster Time elapsed 16.812501907348633
```

```
Cluster1 = LogRegCluster('1')
```

```
Train set: (217, 1065023) (217,)
Test set: (55, 1065023) (55,)
Shape of prediction for training set (217,)
Shape of prediction for test set (55,)
Column of maximum coefficient value is 94838
Maximum coefficient value is 0.31460545719585026
Sorted Coeff values from largest to smallest [[ 94838 152051    668 ... 600681 108485  13894]]
Accuracy Test Score is  0.9090909090909091
Logistic Regression is done for Cluster Time elapsed 17.474863529205322
```

```
Cluster2 = LogRegCluster('2')
```

```
Train set: (217, 1065023) (217,)
Test set: (55, 1065023) (55,)
Shape of prediction for training set (217,)
Shape of prediction for test set (55,)
Column of maximum coefficient value is 13894
Maximum coefficient value is 1.5437219689617128
Sorted Coeff values from largest to smallest [[ 13894 304794 173840 ... 204511 108485  94838]]
Accuracy Test Score is  0.9454545454545454
Logistic Regression is done for Cluster Time elapsed 19.953119039535522
```

```
Cluster3 = LogRegCluster('3')
```

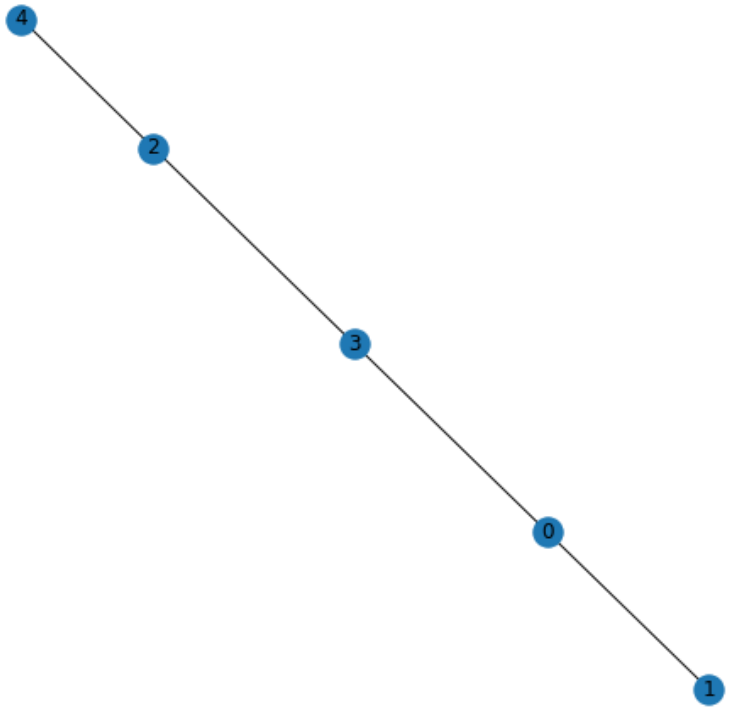
```
Train set: (217, 1065023) (217,)
Test set: (55, 1065023) (55,)
Shape of prediction for training set (217,)
Shape of prediction for test set (55,)
Column of maximum coefficient value is 667
Maximum coefficient value is 1.5536922339238117
Sorted Coeff values from largest to smallest [[   667    668    674 ... 197355  57084  13894]]
Accuracy Test Score is  1.0
Logistic Regression is done for Cluster Time elapsed 19.185578107833862
```

```
Cluster4 = LogRegCluster('4')
```

```
Train set: (217, 1065023) (217,)
Test set: (55, 1065023) (55,)
Shape of prediction for training set (217,)
Shape of prediction for test set (55,)
Column of maximum coefficient value is 204511
Maximum coefficient value is 0.14002161332126623
Sorted Coeff values from largest to smallest [[204511 108485 188237 ... 304794  94838  13894]]
Accuracy Test Score is  0.9272727272727272
Logistic Regression is done for Cluster Time elapsed 19.050964832305908
```

C) To map the five cell clusters to a minimum spanning tree, I first calculated the Manhattan distance between each cluster. The manhattan distance is a better choice than Euclidian in this instance because we are just looking to define the vector distance between each point. Also each cluster center has very high dimensions and the manhattan distance is better for this sparser, more spread out data because it places less emphasis on outliers. Also, since it is the absolute value and not squared like Euclidian it minimizes the noise associated with higher dimensional data.

I then used the distances between each cluster center to weight the corresponding edges in the minimum spanning tree. The minimum spanning tree tells us about how closely related the different clusters are to each other.



Below calculating the manhattan distance between each cluster center.

```
from scipy.spatial import distance

df_man = pd.DataFrame(np.zeros((5,5)))

for i in np.arange(len(y.cluster_centers_)):
    for j in np.arange(len(y.cluster_centers_)):
        mandij = distance.cityblock(y.cluster_centers_[i,:], y.cluster_centers_[j,:])
        df_man.iloc[i,j] = mandij
```

```
df_man
```

	0	1	2	3	4
0	0.000000	0.453060	0.457442	0.399757	0.812639
1	0.453060	0.000000	0.593570	0.627069	0.921011
2	0.457442	0.593570	0.000000	0.371463	0.759257
3	0.399757	0.627069	0.371463	0.000000	0.795173
4	0.812639	0.921011	0.759257	0.795173	0.000000

```
import networkx as nx


fig=plt.figure(figsize=(8,8))

#Generate graph with a center placed for the centroid of each cluster
G = nx.complete_graph(len(np.unique(y.labels_)))
#Calculate the edge length between each cluster using the previously calculated manhattan distance
for u,v in G.edges():
    G[u][v]["weight"]= df_man.iloc[u,v]

    #Generate the minimum spanning tree using Networkx built in function
T=nx.minimum_spanning_tree(G)
pos=nx.spring_layout(T,scale=10000)
nx.draw_networkx(T,pos)
#Draw each edge between the different cluster, weighted by the manhattan distance
edge_labels=dict([((u,v,),round(d['weight'],2))
            for u,v,d in T.edges(data=True)])
plt.axis('off')
```