In [49]:

```python
# (Q1.1) Read the training and testing data.

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


df_Test = pd.read_csv(r'/Users/limshikee/Desktop/test_wbcd.csv')

df_Train = pd.read_csv(r'/Users/limshikee/Desktop/train_wbcd.csv')
```

In [2]:

```python
#(Q1.1) Print the number of features in the dataset.

print ('Number of features:', len(df_Train.columns)-2)
```

```
Number of features: 30
```

In [3]:

```python
#(Q1.1) For the data label, print the total number of B's and M's in the train
ing data.

df_Train['Diagnosis'].value_counts()
```

Out[3]:

```
B    58
M    42
Name: Diagnosis, dtype: int64
```

In [4]:

```python
#(Q1.1) For the data label, print the total number of B's and M's in the testi
ng data.

df_Test['Diagnosis'].value_counts()
```

Out[4]:

```
B    14
M     6
Name: Diagnosis, dtype: int64
```

In [5]:

```
# (Q1.1) Comment on the class distribution. Is it balanced or unbalanced?

print ("The data distribution for class B and M is unbalanced as the ratio of 
M:B for training data is 1:1.38 whereas the ratio of M:B for testing data is 1
:2.33")
```

```
The data distribution for class B and M is unbalanced as the ratio
of M:B for training data is 1:1.38 whereas the ratio of M:B for te
sting data is 1:2.33
```

In [6]:

```
#(Q1.1) Print the number of features with missing entries (feature value is ze
ro)

print ('Total missing entries in the training data:',sum((df_Train == 0).sum()
)+sum(df_Train.isnull().sum()))

print ('Total missing entries in the testing data:',sum((df_Test == 0).sum())+
sum(df_Test.isnull().sum()))
```

```
Total missing entries in the training data: 38
Total missing entries in the testing data: 7
```

In [7]:

```
# (Q1.1) Fill the missing entries. For filling any feature,
# you can use either mean or median value of the feature values from observed
entries.
# Explain the reason behind your choice.

df_Train_New1 = df_Train.replace(to_replace = np.nan, value = df_Train.median(
))
df_Train_New4 = df_Train_New1.replace(to_replace = 0, value = df_Train_New1.me
dian())

print ('Total missing entries in the training data:',sum((df_Train_New4 == 0).
sum())+sum(df_Train_New4.isnull().sum()))

df_Test_New1 = df_Test.replace(to_replace = np.nan, value = df_Test.median())
df_Test_New4 = df_Test_New1.replace(to_replace = 0, value = df_Test_New1.media
n())

print ('Total missing entries in the testing data:',sum((df_Test_New4 == 0).su
m())+sum(df_Test_New4.isnull().sum()))

print ('We use median imputation to replace the missing entries to prevent the
values to be affected by great outliers.')
```

```
Total missing entries in the training data: 0
Total missing entries in the testing data: 0
We use median imputation to replace the missing entries to prevent
the values to be affected by great outliers.
```

In [8]:

```
# (Q1.1) Normalize the training and testing data.

from sklearn.preprocessing import StandardScaler
cols_to_norm = ['f1','f2','f3','f4','f5','f6','f7','f8','f9','f10','f11','f12'
,'f13','f14','f15','f16','f17','f18','f19','f20','f21','f22','f23','f24','f25'
,'f26','f27','f28','f29','f30']
df_Train_New4[cols_to_norm] = StandardScaler().fit_transform(df_Train_New4[col
s_to_norm])

cols_to_norm = ['f1','f2','f3','f4','f5','f6','f7','f8','f9','f10','f11','f12'
,'f13','f14','f15','f16','f17','f18','f19','f20','f21','f22','f23','f24','f25'
,'f26','f27','f28','f29','f30']
df_Test_New4[cols_to_norm] = StandardScaler().fit_transform(df_Test_New4[cols_
to_norm])
```

In [9]:

```
predictors = ['f1','f2','f3','f4','f5','f6','f7','f8','f9','f10','f11','f12','
f13','f14','f15','f16','f17','f18','f19','f20','f21','f22','f23','f24','f25','
f26','f27','f28','f29','f30']
response = ['Diagnosis']
```

In [10]:

```
# (1.2) Train logistic regression models with L1 regularization and L2 regular
ization using alpha = 0.1
# and lambda = 0.1.

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

alpha_val = 0.1
lambda_val = 0.1

# import warnings filter
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)



#Initialize the Logitic regression model with l2 penalty
lr2 = LogisticRegression(C=1/lambda_val, penalty='l2')
lr2.fit(df_Train_New4[predictors], df_Train_New4['Diagnosis'])
```

Out[10]:

```
LogisticRegression(C=10.0, class_weight=None, dual=False, fit_inte
rcept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='wa
rn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [11]:

```
#Initialize the Logitic regression model with l1 penalty
lr1 = LogisticRegression(C=1/alpha_val, penalty='l1')
lr1.fit(df_Train_New4[predictors], df_Train_New4['Diagnosis'])
```

Out[11]:

```
LogisticRegression(C=10.0, class_weight=None, dual=False, fit_inte
rcept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=None, solver='wa
rn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [12]:

```
# (1.2) Report accuracy, precision, recall, f1-score and print the confusion m
atrix (for L1)

from sklearn.metrics import classification_report
from sklearn.metrics import *

y1_predict=lr1.predict(df_Test_New4[predictors])
model_acc = accuracy_score(y1_predict, df_Test_New4['Diagnosis'])
print("L1 Model Accuracy is: {}".format(model_acc))
print(classification_report(df_Test_New4['Diagnosis'], y1_predict))

print ('Confusion Matrix:')
print (confusion_matrix(np.array(df_Test_New4['Diagnosis']),np.array(y1_predic
t)))
```

```
L1 Model Accuracy is: 0.9
              precision    recall  f1-score   support

           B       0.93      0.93      0.93        14
           M       0.83      0.83      0.83         6

   micro avg       0.90      0.90      0.90        20
   macro avg       0.88      0.88      0.88        20
weighted avg       0.90      0.90      0.90        20

Confusion Matrix:
[[13  1]
 [ 1  5]]
```

In [13]:

```
# (1.2) Report accuracy, precision, recall, f1-score and print the confusion m
atrix (for L2)

from sklearn.metrics import classification_report

y2_predict=lr2.predict(df_Test_New4[predictors])
model_acc = accuracy_score(y2_predict, df_Test_New4['Diagnosis'])
print("L2 Model Accuracy is: {}".format(model_acc))
print(classification_report(df_Test_New4['Diagnosis'], y2_predict))

print ('Confusion Matrix:')
print (confusion_matrix(np.array(df_Test_New4['Diagnosis']),np.array(y2_predic
t)))
```

```
L2 Model Accuracy is: 0.9
              precision    recall  f1-score   support

           B       0.93      0.93      0.93        14
           M       0.83      0.83      0.83         6

   micro avg       0.90      0.90      0.90        20
   macro avg       0.88      0.88      0.88        20
weighted avg       0.90      0.90      0.90        20

Confusion Matrix:
[[13  1]
 [ 1  5]]
```

In [14]:

```
fID = 218187754%3

fID
```

Out[14]:

```
1
```

In [15]:

```
#(Q1.3.A) For L1 model, choose the best alpha value from the
# following set: {0.1,1,3,10,33,100,333,1000, 3333, 10000, 33333} based on par
ameter P.
```

In [16]:

```
l1_alpha = [0.1,1,3,10,33,100,333,1000, 3333, 10000, 33333]
```

In [17]:

```
from sklearn.model_selection import train_test_split
```

In [20]:

```
mean_f1a = []
for l1_alpha_val in l1_alpha:
    lr = LogisticRegression(C=1.0/float(l1_alpha_val),penalty='l1')
    print ('Alpha:',l1_alpha_val)
    arr_f1a = []
    for i in range(0,10):
        Dtrain, Dtest = train_test_split(df_Train_New4, test_size=0.3)
        lr.fit(Dtrain[predictors], Dtrain['Diagnosis'])
        y_predict=lr.predict(Dtest[predictors])
        f1a=f1_score(np.array(Dtest['Diagnosis']),np.array(y_predict),labels=n
p.unique(y_predict),average='macro')
        arr_f1a.append(f1a)
    mean_f1a.append(np.mean(arr_f1a))
    print ('F1 Score:',np.mean(arr_f1a))
best_alpha=l1_alpha[np.argmax(mean_f1a)]
print ('best alpha:'+str(l1_alpha[np.argmax(mean_f1a)]))
```

```
Alpha: 0.1
F1 Score: 1.0
Alpha: 1
F1 Score: 0.979022603809544
Alpha: 3
F1 Score: 0.9762862351466357
Alpha: 10
F1 Score: 0.9314937397811981
Alpha: 33
F1 Score: 0.7043404561719859
Alpha: 100
F1 Score: 0.7809939360359528
Alpha: 333
F1 Score: 0.7299500859111732
Alpha: 1000
F1 Score: 0.7327248584311075
Alpha: 3333
F1 Score: 0.764795407524712
Alpha: 10000
F1 Score: 0.7214782415186827
Alpha: 33333
F1 Score: 0.7706803277775921
best alpha:0.1
```

In [71]:

```
#(Q1.3.B) For L2 model, choose the best lambda value from the
# following set: {0.001, 0.003, 0.01, 0.03, 0.1,0.3,1,3,10,33} based on parame
ter P.
```

In [22]:

```
l2_lambda = [0.001, 0.003, 0.01, 0.03, 0.1,0.3,1,3,10,33]
```

In [23]:

```
mean_f1 = []
for l2_lambda_val in l2_lambda:
    lr = LogisticRegression(C=1.0/float(l2_lambda_val),penalty='l2')
    print ('lambda:',l2_lambda_val)
    arr_f1 = []
    for i in range(0,10):
        Dtrain, Dtest = train_test_split(df_Train_New4, test_size=0.3)
        lr.fit(Dtrain[predictors], Dtrain['Diagnosis'])
        y_predict=lr.predict(Dtest[predictors])
        f1=f1_score(np.array(Dtest['Diagnosis']),np.array(y_predict),labels=np
.unique(y_predict),average='macro')
        arr_f1.append(f1)
    mean_f1.append(np.mean(arr_f1))
    print ('F1 Score:',np.mean(arr_f1))
best_lambda = l2_lambda[np.argmax(mean_f1)]
print ('best lambda:'+str(l2_lambda[np.argmax(mean_f1)]))
```

```
lambda: 0.001
F1 Score: 0.9832232639440649
lambda: 0.003
F1 Score: 1.0
lambda: 0.01
F1 Score: 0.9932044252044252
lambda: 0.03
F1 Score: 0.9863503641315873
lambda: 0.1
F1 Score: 0.9961489088575096
lambda: 0.3
F1 Score: 0.9966329966329965
lambda: 1
F1 Score: 0.996662958843159
lambda: 3
F1 Score: 0.9929999551801554
lambda: 10
F1 Score: 0.9729150030952034
lambda: 33
F1 Score: 0.9858374610460618
best lambda:0.003
```

In [74]:

```
#(Q1.3.C) Use the best alpha and lambda parameter to re-train your final L1 an
d L2 regularized model.
# Evaluate the prediction performance on the test data and report the followin
g:
# •Precision and Accuracy
# •The top 5 features selected in decreasing order of feature weights.
# •Confusion matrix
```

In [26]:

```
#Final L1 Model
lrfinal = LogisticRegression(C=1.0/float(best_alpha),penalty='l1')
lrfinal.fit(df_Train_New4[predictors], df_Train_New4['Diagnosis'])
```

Out[26]:

```
LogisticRegression(C=10.0, class_weight=None, dual=False, fit_inte
rcept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=None, solver='wa
rn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [27]:

```
y1final_predict=lrfinal.predict(df_Test_New4[predictors])
model_acc = accuracy_score(y1final_predict, df_Test_New4['Diagnosis'])
print("L1 Model Accuracy is: {}".format(model_acc))
print(classification_report(df_Test_New4['Diagnosis'], y1final_predict))

print ('Confusion Matrix:')
print (confusion_matrix(np.array(df_Test_New4['Diagnosis']),np.array(y1final_p
redict)))
```

```
L1 Model Accuracy is: 0.9
              precision    recall  f1-score   support

           B       0.93      0.93      0.93        14
           M       0.83      0.83      0.83         6

   micro avg       0.90      0.90      0.90        20
   macro avg       0.88      0.88      0.88        20
weighted avg       0.90      0.90      0.90        20

Confusion Matrix:
[[13  1]
 [ 1  5]]
```

In [28]:

```
#Final L2 Model
lrfinal2 = LogisticRegression(C=1.0/float(best_lambda),penalty='l2')
lrfinal2.fit(df_Train_New4[predictors], df_Train_New4['Diagnosis'])
```

Out[28]:

```
LogisticRegression(C=333.3333333333333, class_weight=None, dual=Fa
lse,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l2', random_st
ate=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
```

In [29]:

```
y2final_predict=lrfinal2.predict(df_Test_New4[predictors])
model_acc = accuracy_score(y2final_predict, df_Test_New4['Diagnosis'])
print("L2 Model Accuracy is: {}".format(model_acc))
print(classification_report(df_Test_New4['Diagnosis'], y2final_predict))

print ('Confusion Matrix:')
print (confusion_matrix(np.array(df_Test_New4['Diagnosis']),np.array(y2final_p
redict)))
```

```
L2 Model Accuracy is: 0.85
              precision    recall  f1-score   support

           B       0.92      0.86      0.89        14
           M       0.71      0.83      0.77         6

   micro avg       0.85      0.85      0.85        20
   macro avg       0.82      0.85      0.83        20
weighted avg       0.86      0.85      0.85        20

Confusion Matrix:
[[12  2]
 [ 1  5]]
```

In [50]:

```
#(Q2.1.1) Use the data from the file reduced_mnist.csv in the data directory.
# Begin by reading the data.
# Print the following information: •Number of data points •Total number of fea
tures •Unique labels in the data

df_mnist = pd.read_csv(r'/Users/limshikee/Desktop/reduced_mnist.csv')
#print(df_mnist)
```

In [34]:

```
print ('Number of data points:',len(df_mnist))
print ('Total number of features:',len(df_mnist.columns)-2)
print ('Unique labels in the data:',np.unique(np.array(df_mnist.label)))
```

```
Number of data points: 2520
Total number of features: 783
Unique labels in the data: [0 1 2 3 4 5 6 7 8 9]
```

In [35]:

```
#(Q2.1.2) Split the data into 70% training data and 30% test data.
# Fit a One-vs-Rest Classifier (which uses Logistic regression classifier with
alpha=1) on training data,
# and report accuracy, precision, recall on testing data.
```

In [36]:

```
alpha = 1

mnist_label = df_mnist[['label']]

mnist_predictors = df_mnist.drop('label', axis=1)

final_predictors = mnist_predictors.columns.values.tolist()
final_label = mnist_label.columns.values.tolist()
```

In [37]:

```
Q2Dtrain,Q2Dtest = train_test_split(df_mnist, test_size=0.3)


lr1_Q2 = LogisticRegression(C=1/alpha, penalty='l1')
lr1_Q2.fit(Q2Dtrain[final_predictors], Q2Dtrain['label'])
```

Out[37]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_inter
cept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=None, solver='wa
rn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [38]:

```
Q2_y_predict=lr1_Q2.predict(Q2Dtest[final_predictors])
model_acc_Q2 = accuracy_score(Q2_y_predict, Q2Dtest['label'])
print("Q2 Model Accuracy is: {}".format(model_acc_Q2))
print(classification_report(Q2Dtest['label'], Q2_y_predict))

print ('Confusion Matrix:')
print (confusion_matrix(np.array(Q2Dtest['label']),np.array(Q2_y_predict)))
```

```
Q2 Model Accuracy is: 0.8492063492063492
              precision    recall  f1-score   support

           0       0.86      0.95      0.91        66
           1       0.91      0.99      0.95        84
           2       0.86      0.79      0.82        80
           3       0.80      0.80      0.80        71
           4       0.85      0.88      0.86        81
           5       0.83      0.72      0.78        69
           6       0.89      0.95      0.92        66
           7       0.88      0.83      0.86        82
           8       0.72      0.73      0.73        71
           9       0.86      0.84      0.85        86

   micro avg       0.85      0.85      0.85       756
   macro avg       0.85      0.85      0.85       756
weighted avg       0.85      0.85      0.85       756

Confusion Matrix:
[[63  0  0  1  0  0  1  0  1  0]
 [ 0 83  0  0  0  0  0  0  1  0]
 [ 1  1 63  1  1  1  4  2  6  0]
 [ 3  0  3 57  0  6  0  0  2  0]
 [ 0  0  0  0 71  0  0  1  5  4]
 [ 4  1  0  7  1 50  0  1  3  2]
 [ 1  0  1  0  1  0 63  0  0  0]
 [ 0  0  4  0  2  1  1 68  1  5]
 [ 1  6  1  5  1  2  2  0 52  1]
 [ 0  0  1  0  7  0  0  5  1 72]]
```

In [39]:

```
#(Q2.2.1) Choose the best value of alpha from the set a={0.1, 1, 3, 10, 33, 100, 333, 1000, 3333, 10000, 33333}
# by observing average training and validation performance P.
```

In [40]:

```
a = [0.1, 1, 3, 10, 33, 100, 333, 1000, 3333, 10000, 33333]
```

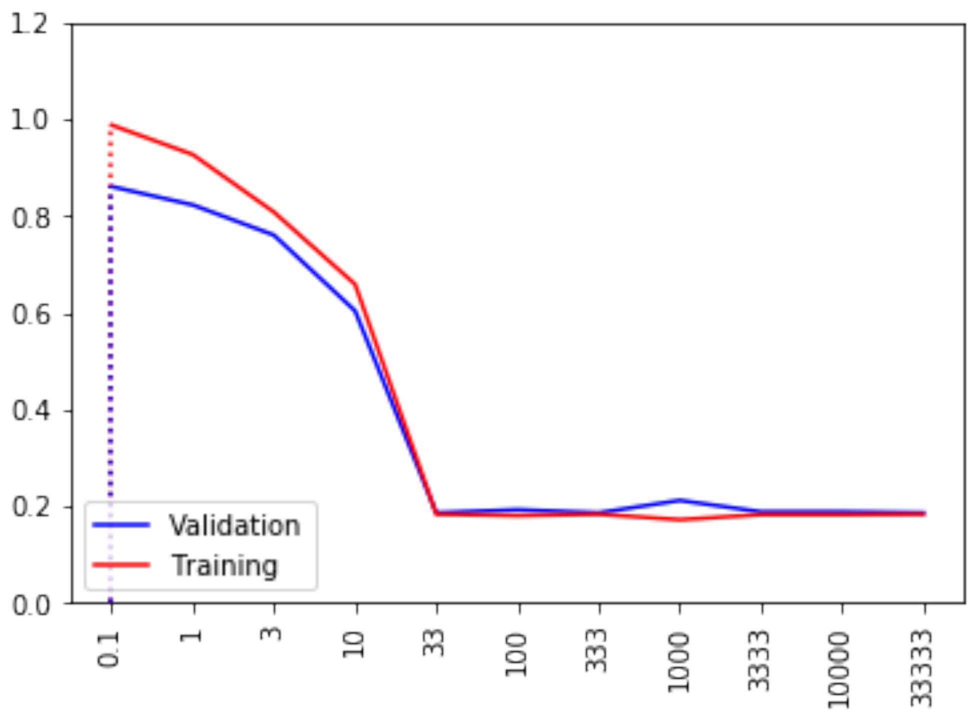
In [43]:

```
Q2_test_f1 = []
Q2_train_f1 = []
for l1_alpha_val in a:
    Q2_lr_G = LogisticRegression(tol = l1_alpha_val, C=1.0,penalty='l1')
    #Q2_lr_G = LogisticRegression(C=1.0/l1_alpha_val,penalty='l1')
    print ('Alpha:',l1_alpha_val)
    Q2_arr_f1 = []
    Q2_arr_f12 = []
    for i in range(0,10):
        Q2_Dtrain, Q2_Dtest = train_test_split(df_mnist, test_size=0.3)
        Q2_lr_G.fit(Q2_Dtrain[final_predictors], Q2_Dtrain['label'])
        Q2_y2_predict=Q2_lr_G.predict(Q2_Dtest[final_predictors])
        Q2_y2_train_predict=Q2_lr_G.predict(Q2_Dtrain[final_predictors])
        Q2_Test_f1=f1_score(np.array(Q2_Dtest['label']),np.array(Q2_y2_predict
),labels=np.unique(Q2_y2_predict),average='macro')
        Q2_arr_f1.append(Q2_Test_f1)
        Q2_Train_f1=f1_score(np.array(Q2_Dtrain['label']),np.array(Q2_y2_train
_predict),labels=np.unique(Q2_y2_train_predict),average='macro')
        Q2_arr_f12.append(Q2_Train_f1)
    Q2_test_f1.append(np.mean(Q2_Test_f1))
    Q2_train_f1.append(np.mean(Q2_Train_f1))
    print ('F1 Score_Test:',np.mean(Q2_Test_f1))
    print ('F1 Score_Train:',np.mean(Q2_Train_f1))
print ('Q2 best alpha for training:'+str(a[np.argmax(Q2_train_f1)]))
print ('Q2 best alpha for validation:'+str(a[np.argmax(Q2_test_f1)]))
Q2_best_alph_training=a[np.argmax(Q2_train_f1)]
Q2_best_alph_validation=a[np.argmax(Q2_test_f1)]
max_index_train  = np.argmax(Q2_train_f1)
max_index_validation  = np.argmax(Q2_test_f1)
```

```
Alpha: 0.1
F1 Score_Test: 0.8615789851043714
F1 Score_Train: 0.9886628147431218
Alpha: 1
F1 Score_Test: 0.8236033120821988
F1 Score_Train: 0.9277636239102842
Alpha: 3
F1 Score_Test: 0.7611947561522673
F1 Score_Train: 0.80880959107851
Alpha: 10
F1 Score_Test: 0.6046175967515672
F1 Score_Train: 0.6595377703807328
Alpha: 33
F1 Score_Test: 0.1870503597122302
F1 Score_Train: 0.18425115800308803
Alpha: 100
F1 Score_Test: 0.19354838709677416
F1 Score_Train: 0.1814432989690722
Alpha: 333
F1 Score_Test: 0.1870503597122302
F1 Score_Train: 0.18425115800308803
Alpha: 1000
F1 Score_Test: 0.21276595744680848
F1 Score_Train: 0.17296737441740032
Alpha: 3333
F1 Score_Test: 0.18922155688622755
F1 Score_Train: 0.18331616889804325
Alpha: 10000
F1 Score_Test: 0.18922155688622755
F1 Score_Train: 0.18331616889804325
Alpha: 33333
F1 Score_Test: 0.1870503597122302
F1 Score_Train: 0.18425115800308803
Q2 best alpha for training:0.1
Q2 best alpha for validation:0.1
```

In [48]:

```
plt.plot(range(0,len(a)), Q2_test_f1, color='b', label='Validation')
plt.plot(range(0,len(a)), Q2_train_f1, color='r', label='Training')

plt.xticks(range(0,len(a)), a, rotation='vertical')

plt.plot((max_index_train, max_index_train), (0, Q2_train_f1[max_index_train]), ls='dotted', color='r')
plt.plot((max_index_validation, max_index_validation), (0, Q2_test_f1[max_index_validation]), ls='dotted', color='b')

axes = plt.gca()
axes.set_ylim([0, 1.2])

plt.legend(loc="lower left")
plt.show()

print ("Overfitting happens when alpha is less 33 whereas underfitting happens when alpha is greater than 33")
```




```
Overfitting happens when alpha is less 33 whereas underfitting happens when alpha is greater than 33
```

In [45]:

```
#(Q2.2.2) Use the best alpha and all training data to build the final model and
# then evaluate the prediction performance on test data and report the following:
# •The confusion matrix
# •Precision, recall and accuracy for each class
```

In [46]:

```
lr1_Q2_final = LogisticRegression(tol = Q2_best_alph_training, C=1, penalty='l
1')
lr1_Q2_final.fit(Q2Dtrain[final_predictors], Q2Dtrain['label'])

Q2_y_predict_final=lr1_Q2_final.predict(Q2Dtest[final_predictors])
model_acc_Q2_final = accuracy_score(Q2_y_predict_final, Q2Dtest['label'])
print("Q2 Model Accuracy is: {}".format(model_acc_Q2_final))
print(classification_report(Q2Dtest['label'], Q2_y_predict_final))

print ('Confusion Matrix:')
print (confusion_matrix(np.array(Q2Dtest['label']),np.array(Q2_y_predict_final
)))
```

```
Q2 Model Accuracy is: 0.8518518518518519
              precision    recall  f1-score   support

           0       0.90      0.95      0.93        66
           1       0.86      0.99      0.92        84
           2       0.86      0.78      0.82        80
           3       0.76      0.83      0.79        71
           4       0.85      0.86      0.86        81
           5       0.88      0.72      0.79        69
           6       0.91      0.94      0.93        66
           7       0.89      0.83      0.86        82
           8       0.80      0.75      0.77        71
           9       0.82      0.86      0.84        86

   micro avg       0.85      0.85      0.85       756
   macro avg       0.85      0.85      0.85       756
weighted avg       0.85      0.85      0.85       756

Confusion Matrix:
[[63  0  1  0  0  0  2  0  0  0]
 [ 0 83  0  1  0  0  0  0  0  0]
 [ 0  2 62  4  2  1  1  2  6  0]
 [ 2  2  4 59  0  1  0  1  2  0]
 [ 0  1  0  1 70  0  1  1  1  6]
 [ 2  0  1  8  1 50  1  1  3  2]
 [ 1  0  0  0  2  1 62  0  0  0]
 [ 0  4  3  0  2  0  0 68  0  5]
 [ 2  5  1  4  0  2  1  0 53  3]
 [ 0  0  0  1  5  2  0  3  1 74]]
```

In [ ]: