MNIST Dataset - Machine Learning

Below, We import all neccessary libraries for reading data and machine learning.

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, (
from sklearn import metrics
import seaborn as sns
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt
from sklearn.multiclass import OneVsRestClassifier
```

Read the train and test dataframes using pandas. NOTE: .txt files were converted to .csv files for convenience.

In [2]:
```
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

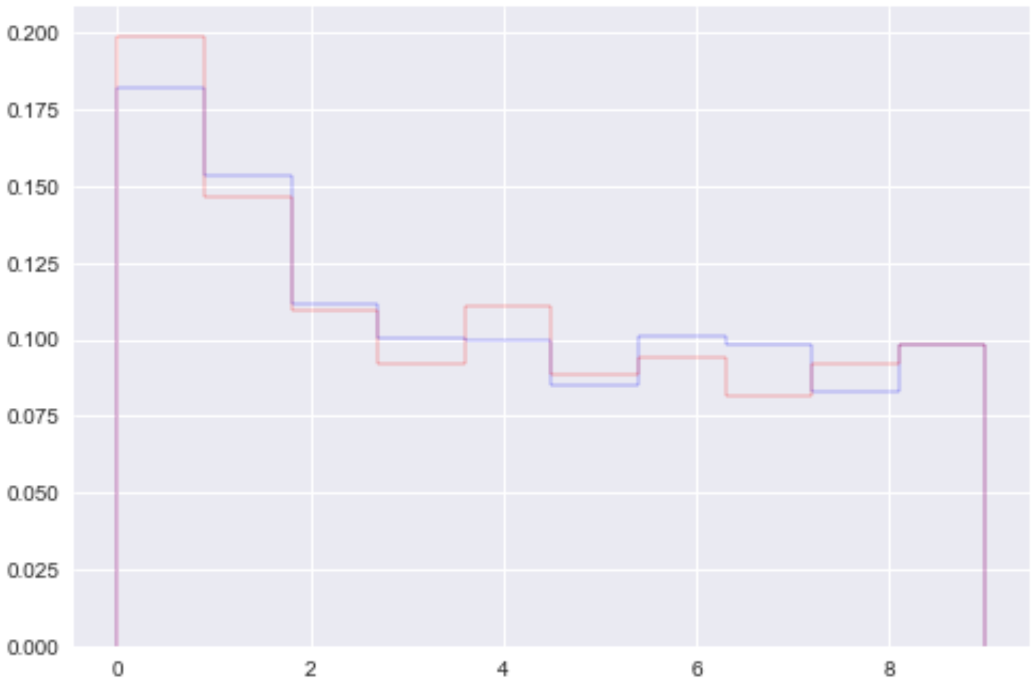
Below, the train and test dataframes are separated into 256 features and 1 class label.

In [3]:
```
train_y = np.array(train['V1'])
del train['V1']
train_x = np.array(train)

test_y = np.array(test['V1'])
del test['V1']
test_x = np.array(test)
```

In [4]:
```
n_classes = 10
```

The graph below shows that the relative proportions for all digits 0..9 are approximately the same in the training and test dataset. This is good since there is no imbalance between the two datasets.

In [5]:
```
plt.hist(train_y, normed = True, histtype='step', color = 'blue')
plt.hist(test_y, normed = True, histtype='step', color = 'red')
plt.show()
```



Initialising an instance of LogisticRegression classifier in sklearn library.

In [6]:
```
LR = LogisticRegression()
LR.fit(train_x, train_y)
```

Out[6]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_interce
pt=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jo
bs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0
.0001,
          verbose=0, warm_start=False)
```

Evaluating our model on the test dataset. We get a test accuracy of 91.1%.

In [7]:
```
score_LR = LR.score(test_x, test_y)
print score_LR
```

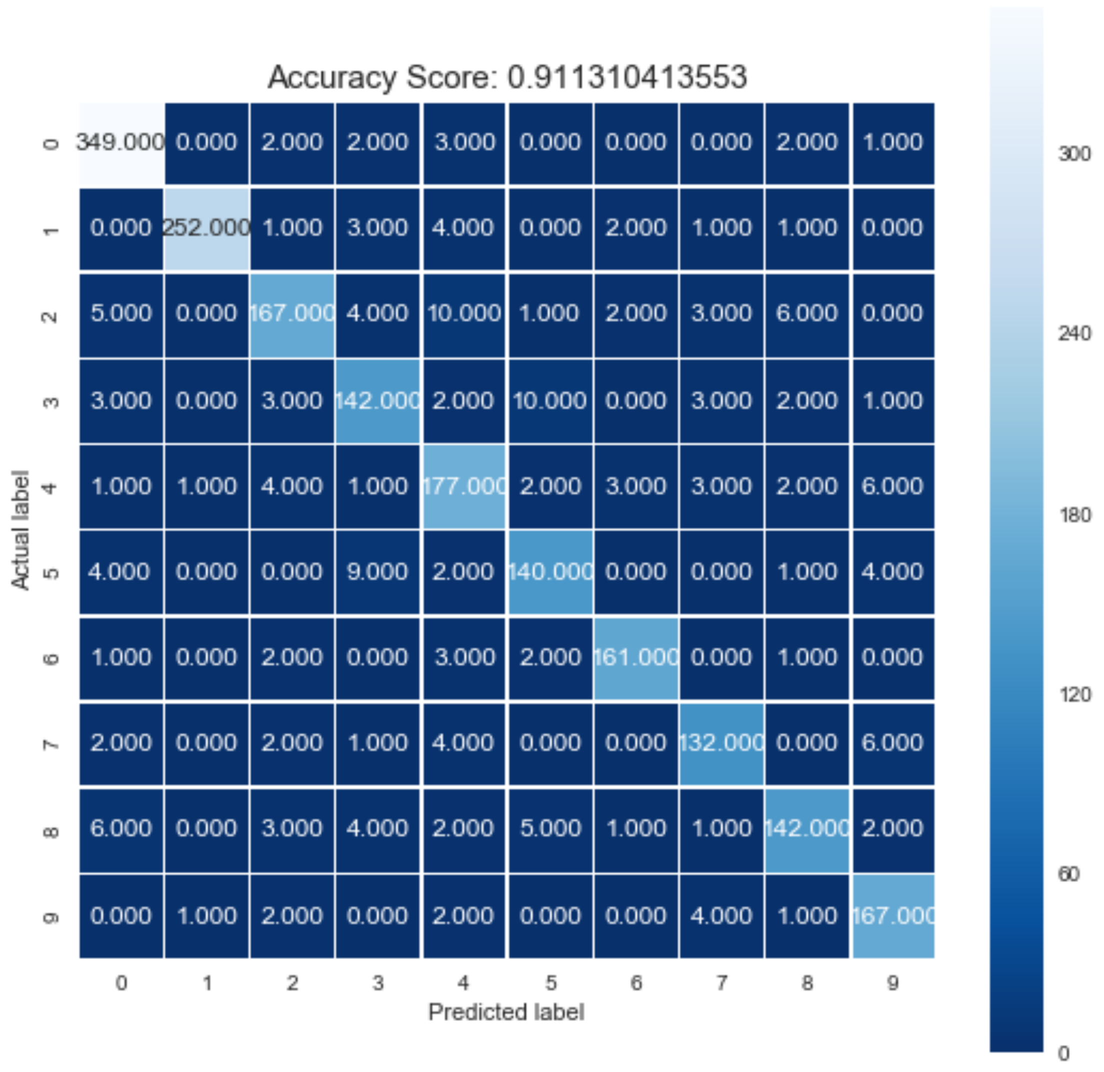
```
0.911310413553
```

In [8]:
```
predictions_LR = LR.predict(test_x)
```

Plotting the confusion matrix for LogisticRegression classifier using seaborn library.

As we can see, the classifier has a fairly high accuracy. However, it is difficult to understand the performance for each digit using this confusion matrix.

In [9]:
```
cm_LR = metrics.confusion_matrix(test_y, predictions_LR)
plt.figure(figsize=(9,9))
sns.heatmap(cm_LR, annot=True, fmt=".3f", linewidths=.5, square = True
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(score_LR)
plt.title(all_sample_title, size = 15);
plt.show()
```



Initialising an instance of LDA classifier, training and evaluating on test data. We see that the accuracy on test data drops to 88.5, which is lower than LogisticRegression.

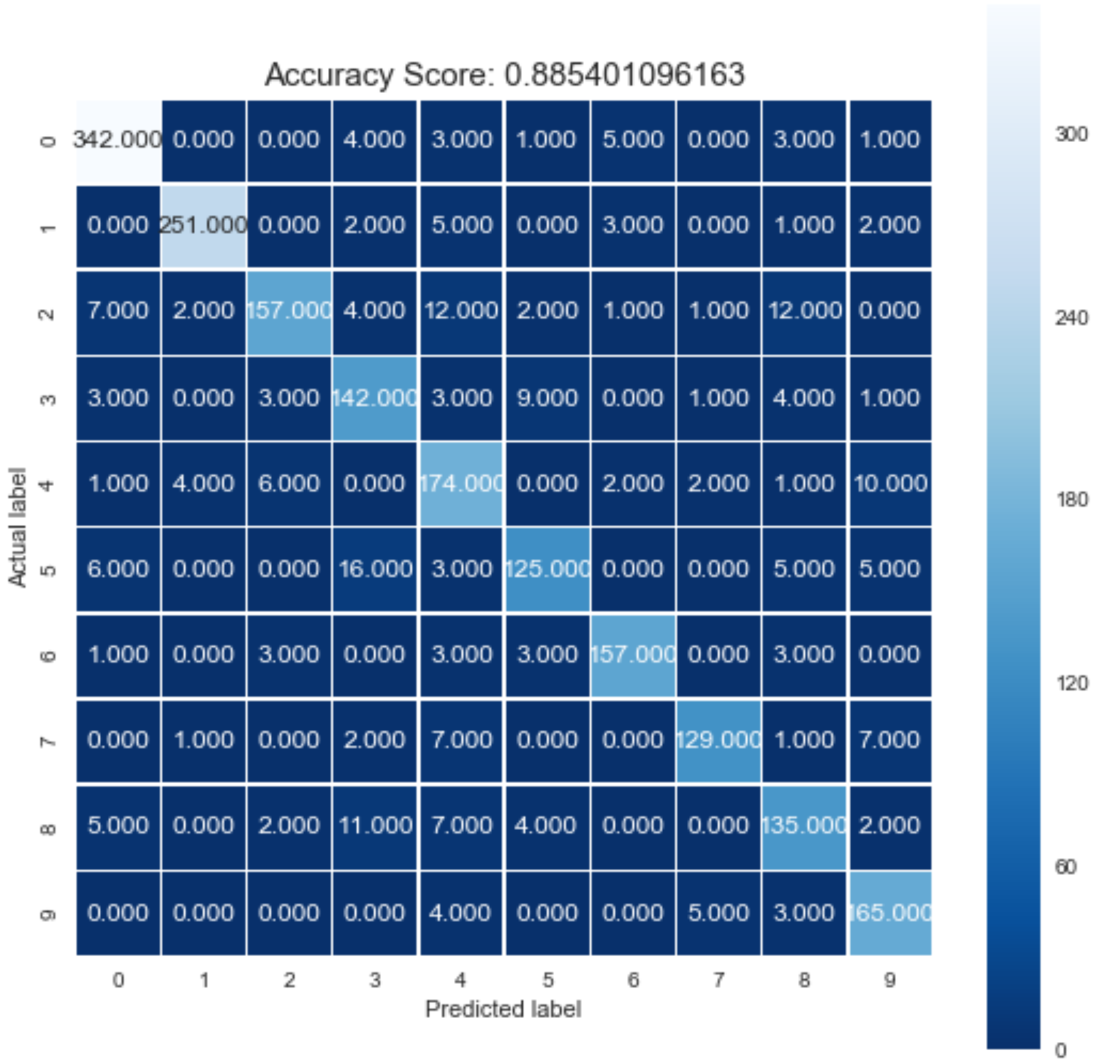
In [10]:
```
LDA = LinearDiscriminantAnalysis()
LDA.fit(train_x, train_y)
score_LDA = LDA.score(test_x, test_y)
print score_LDA
```

```
0.885401096163
```

In [11]:
```
predictions_LDA = LDA.predict(test_x)
cm_LDA = metrics.confusion_matrix(test_y, predictions_LDA)
```

Visualising the confusion matrix tells us that the accuracy is lower. In particular, LDA performs significantly worse on the digit 5.

In [12]:
```
plt.figure(figsize=(9,9))
sns.heatmap(cm_LDA, annot=True, fmt=".3f", linewidths=.5, square = Tru
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score: {0}'.format(score_LDA)
plt.title(all_sample_title, size = 15)
plt.show()
```



Similar analysis for QuadraticDiscriminantAnalysis. Accuracy is 81.3%

In [13]:
```
QDA = QuadraticDiscriminantAnalysis()
QDA.fit(train_x, train_y)
score_QDA = QDA.score(test_x, test_y)
print score_QDA
```

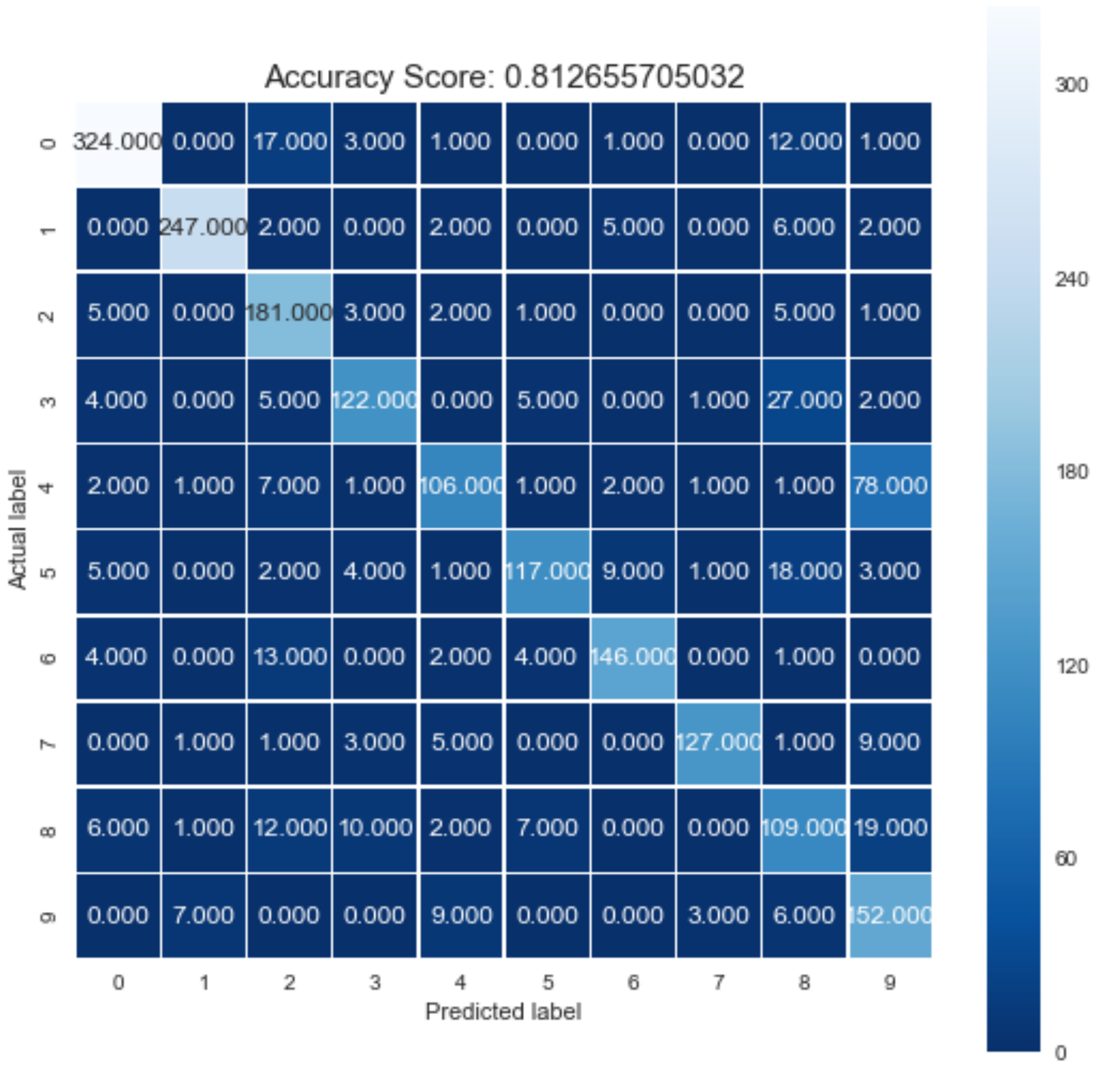
```
/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site
-packages/sklearn/discriminant_analysis.py:695: UserWarning: Variabl
es are collinear
  warnings.warn("Variables are collinear")
```

```
0.812655705032
```

In [14]:
```
predictions_QDA = QDA.predict(test_x)
cm_QDA = metrics.confusion_matrix(test_y, predictions_QDA)
```

Visualising the confusion matrix tells us that the accuracy is lower. In particular, QDA performs significantly worse on the digits 3,5 and 8. Also, it performs better than both LDA and LogisticRegression on the digit 2.

In [15]:
```
plt.figure(figsize=(9,9))
sns.heatmap(cm_QDA, annot=True, fmt=".3f", linewidths=.5, square = Tru
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score: {0}'.format(score_QDA)
plt.title(all_sample_title, size = 15)
plt.show()
```



	0	1	2	3	4	5	6	7	8	9
0	324.000	0.000	17.000	3.000	1.000	0.000	1.000	0.000	12.000	1.000
1	0.000	247.000	2.000	0.000	2.000	0.000	5.000	0.000	6.000	2.000
2	5.000	0.000	181.000	3.000	2.000	1.000	0.000	0.000	5.000	1.000
3	4.000	0.000	5.000	122.000	0.000	5.000	0.000	1.000	27.000	2.000
4	2.000	1.000	7.000	1.000	106.000	1.000	2.000	1.000	1.000	78.000
5	5.000	0.000	2.000	4.000	1.000	117.000	9.000	1.000	18.000	3.000
6	4.000	0.000	13.000	0.000	2.000	4.000	146.000	0.000	1.000	0.000
7	0.000	1.000	1.000	3.000	5.000	0.000	0.000	127.000	1.000	9.000
8	6.000	1.000	12.000	10.000	2.000	7.000	0.000	0.000	109.000	19.000
9	0.000	7.000	0.000	0.000	9.000	0.000	0.000	3.000	6.000	152.000

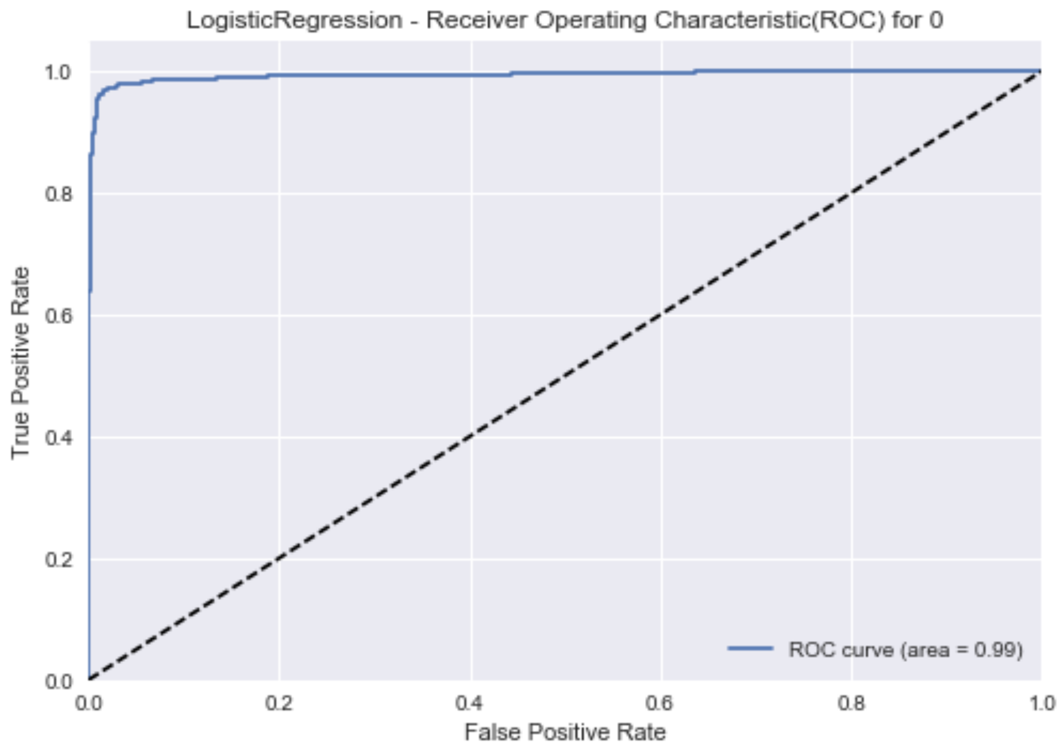
To visualise the results in a better way and get an idea of TP/FP for each digit, we run the same algorithms using a multiclass (OnevsAll classification) approach. Our aim is to plot ROC curves for each digit.

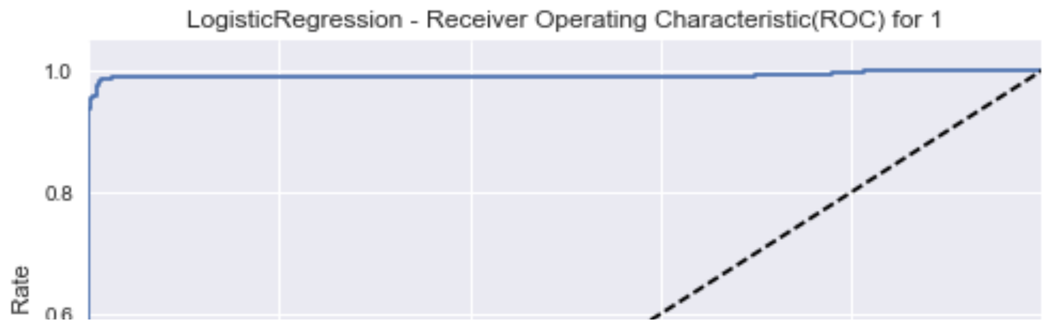
In [16]:
```
train_y = label_binarize(train_y, classes=[0,1,2,3,4,5,6,7,8,9])
test_y = label_binarize(test_y, classes=[0,1,2,3,4,5,6,7,8,9])
```

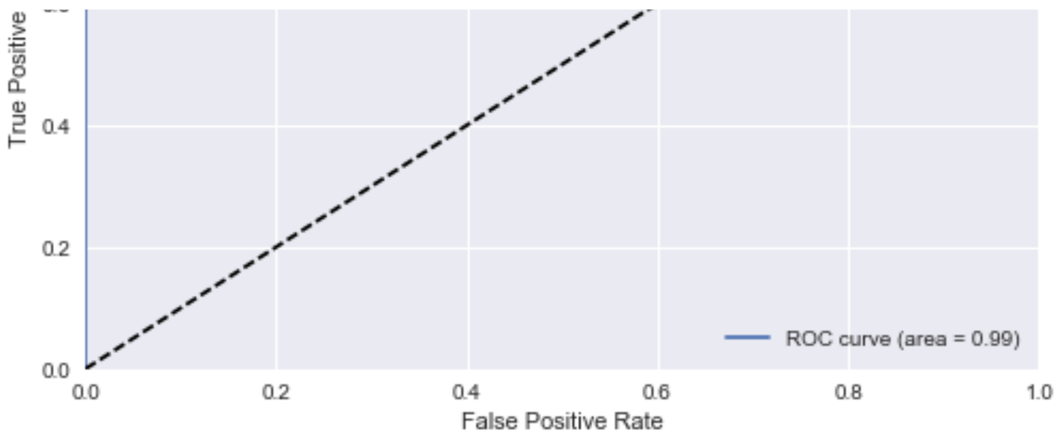
In [17]:
```
LR = OneVsRestClassifier(LogisticRegression())
y_score = LR.fit(train_x, train_y).decision_function(test_x)
```

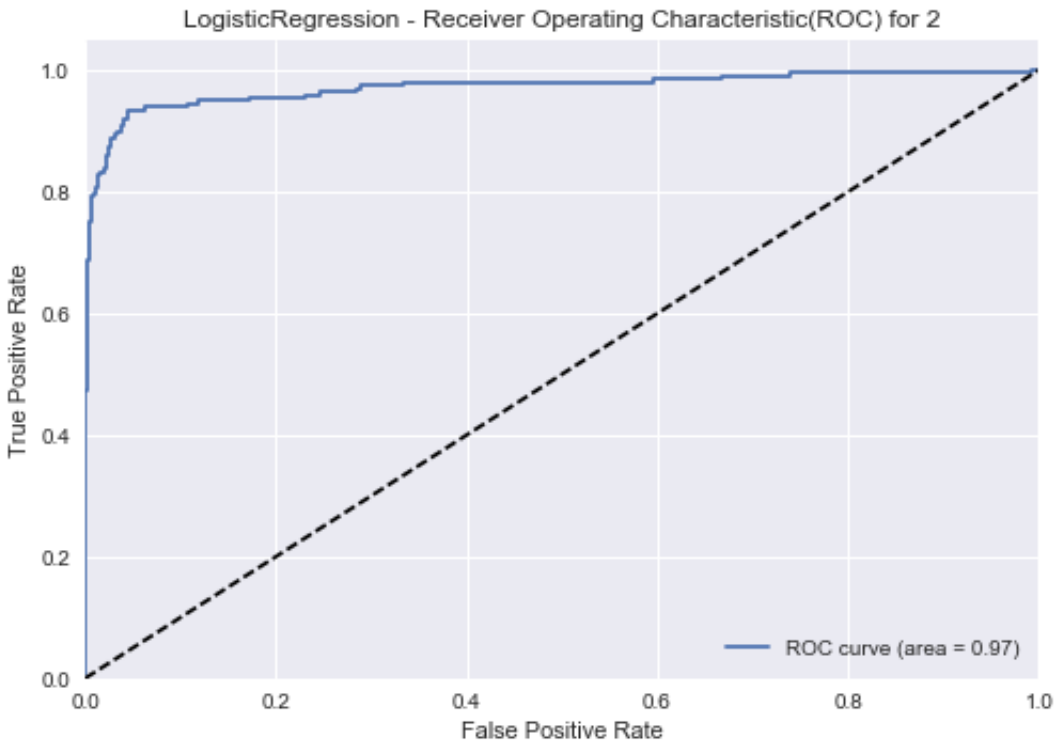
The ROC curves show that LogisticRegression peforms best for the digit 6 giving an area of 1.0.

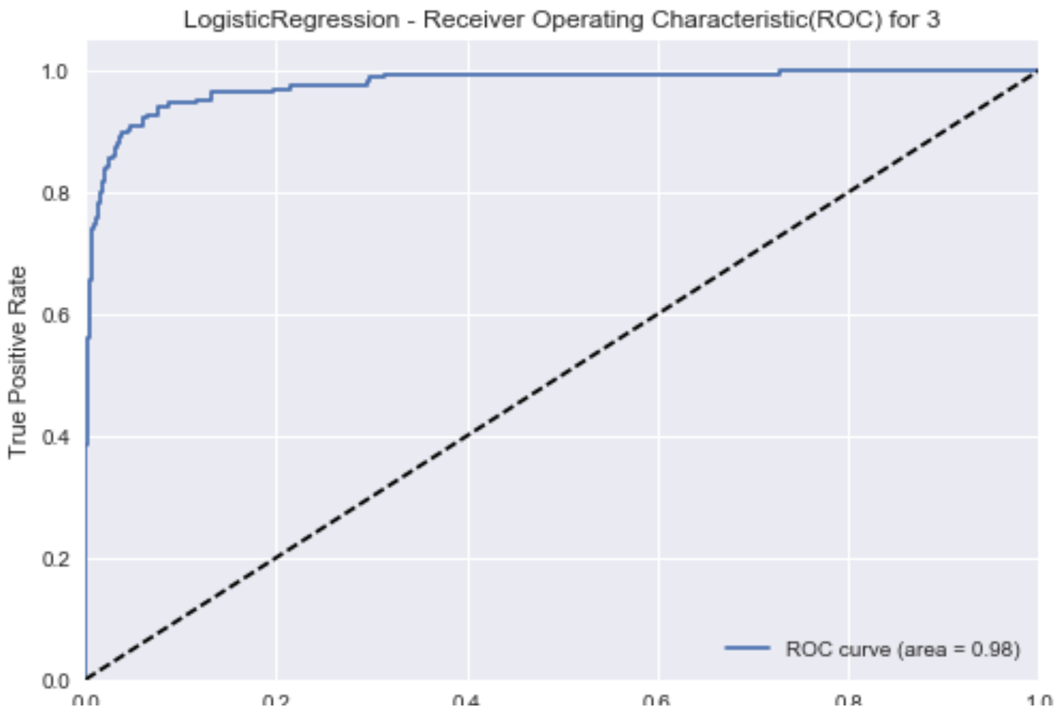
In [18]:
```
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = metrics.roc_curve(test_y[:, i], y_score[:, i])
    roc_auc[i] = metrics.auc(fpr[i], tpr[i])

for i in range(n_classes):
    plt.figure()
    plt.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f)' % roc_au
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('LogisticRegression - Receiver Operating Characteristic(
    plt.legend(loc="lower right")
    plt.show()
```



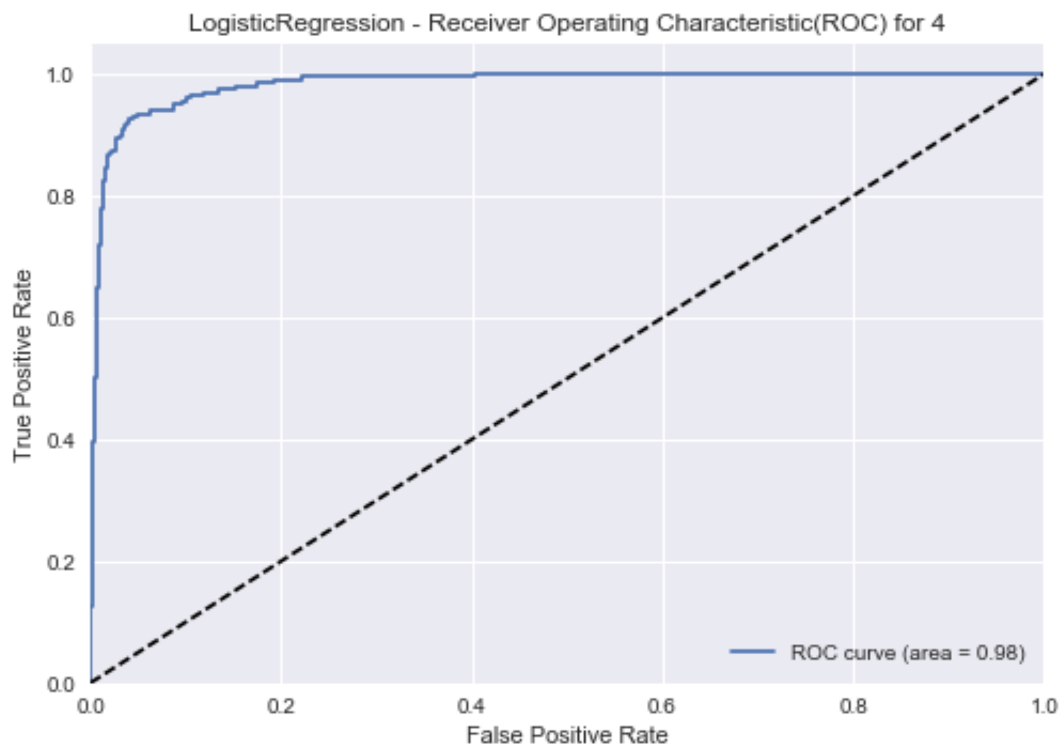


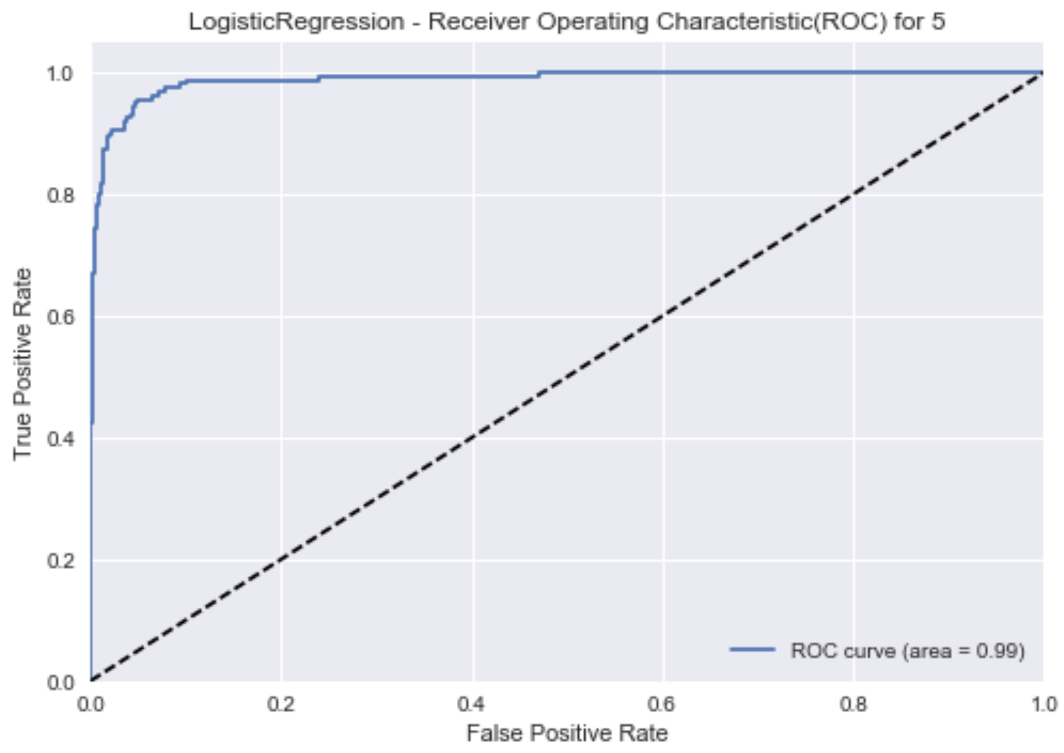

True Positive
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
ROC curve (area = 0.99)

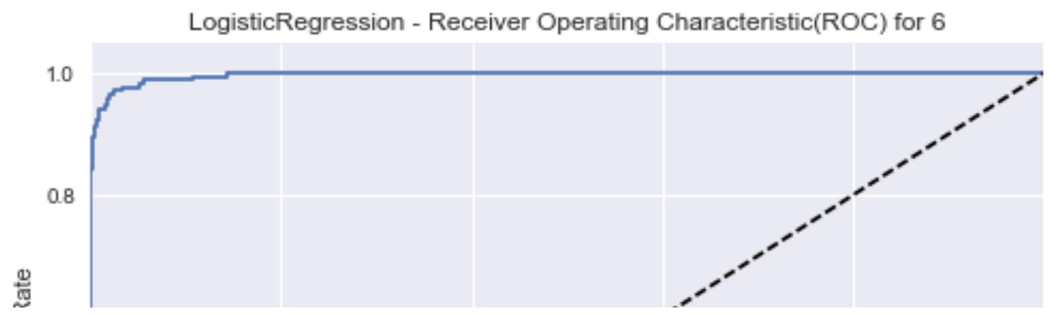

LogisticRegression - Receiver Operating Characteristic(ROC) for 2
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
ROC curve (area = 0.97)

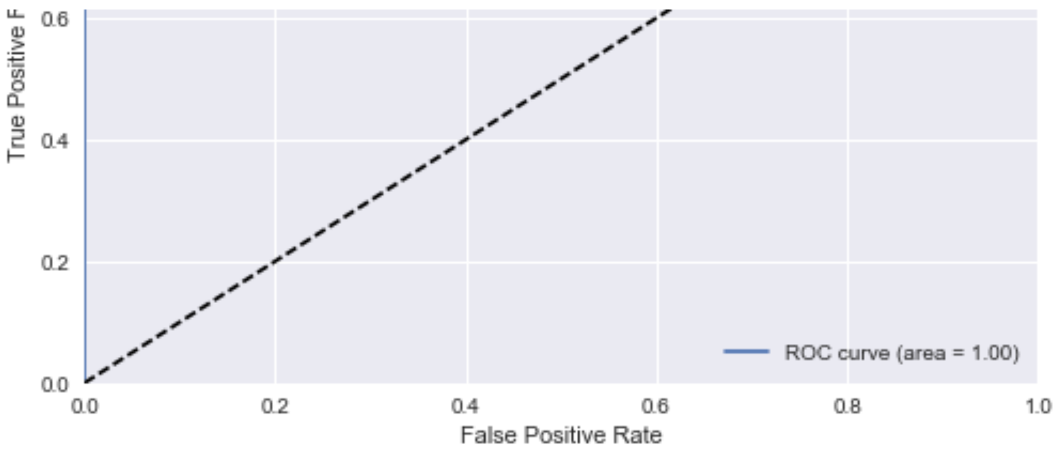

LogisticRegression - Receiver Operating Characteristic(ROC) for 3
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
ROC curve (area = 0.98)

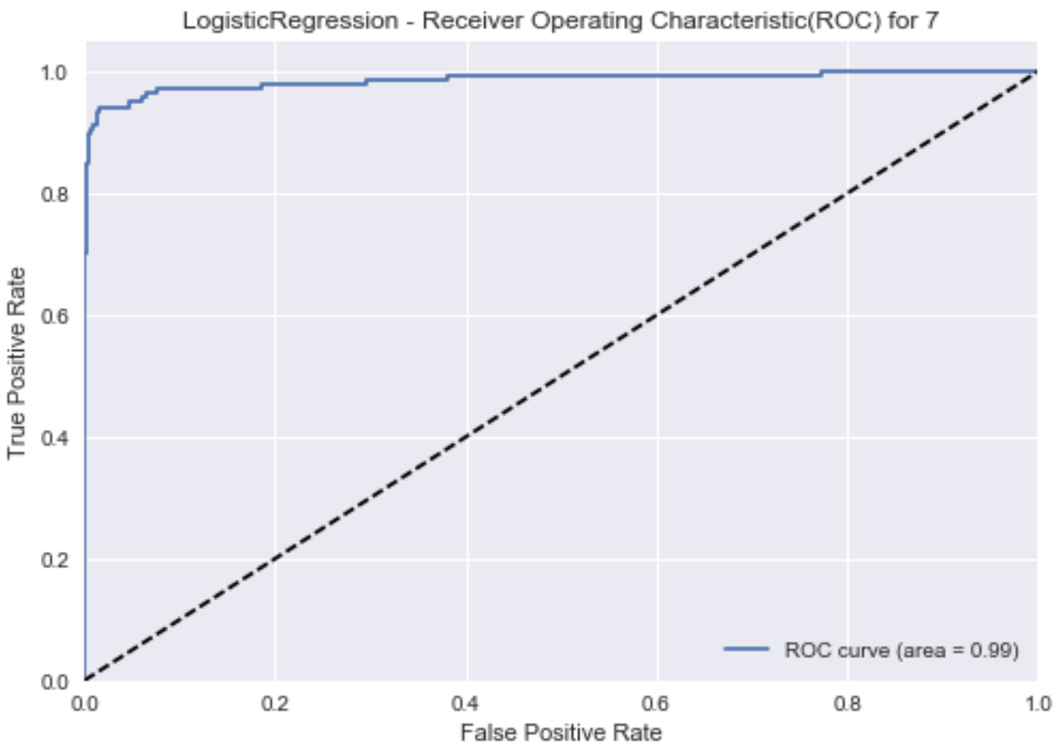
False Positive Rate

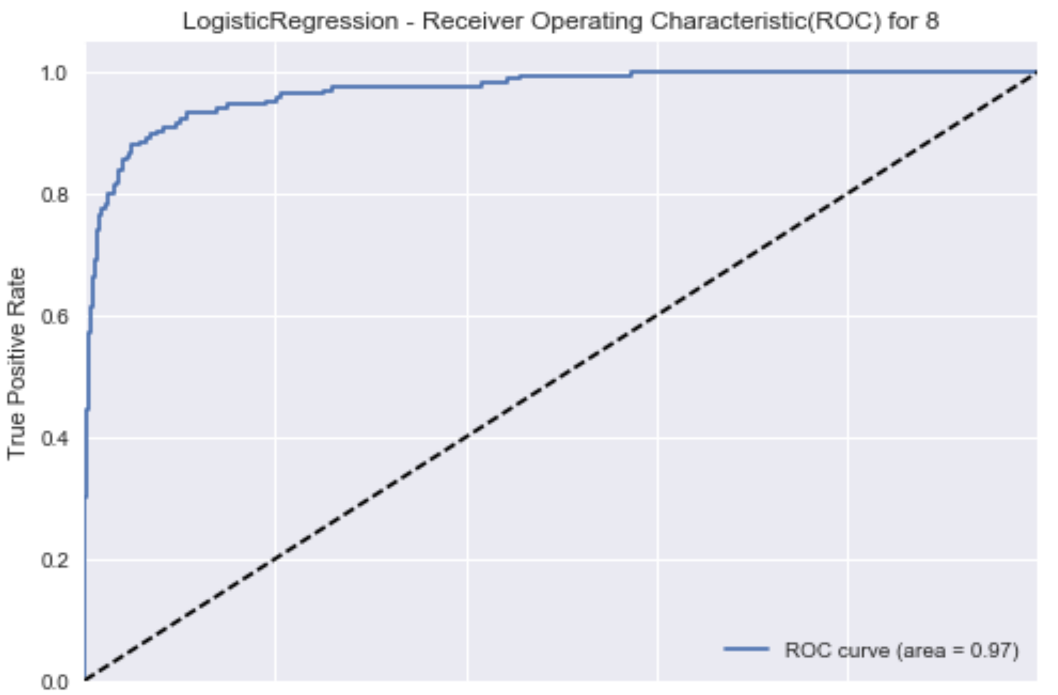

LogisticRegression - Receiver Operating Characteristic(ROC) for 4
True Positive Rate
False Positive Rate
ROC curve (area = 0.98)
0.0
0.2
0.4
0.6
0.8
1.0

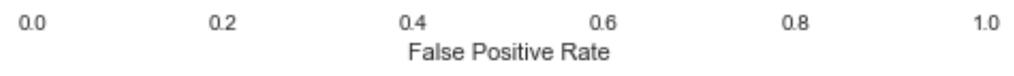

LogisticRegression - Receiver Operating Characteristic(ROC) for 5
True Positive Rate
False Positive Rate
ROC curve (area = 0.99)
0.0
0.2
0.4
0.6
0.8
1.0

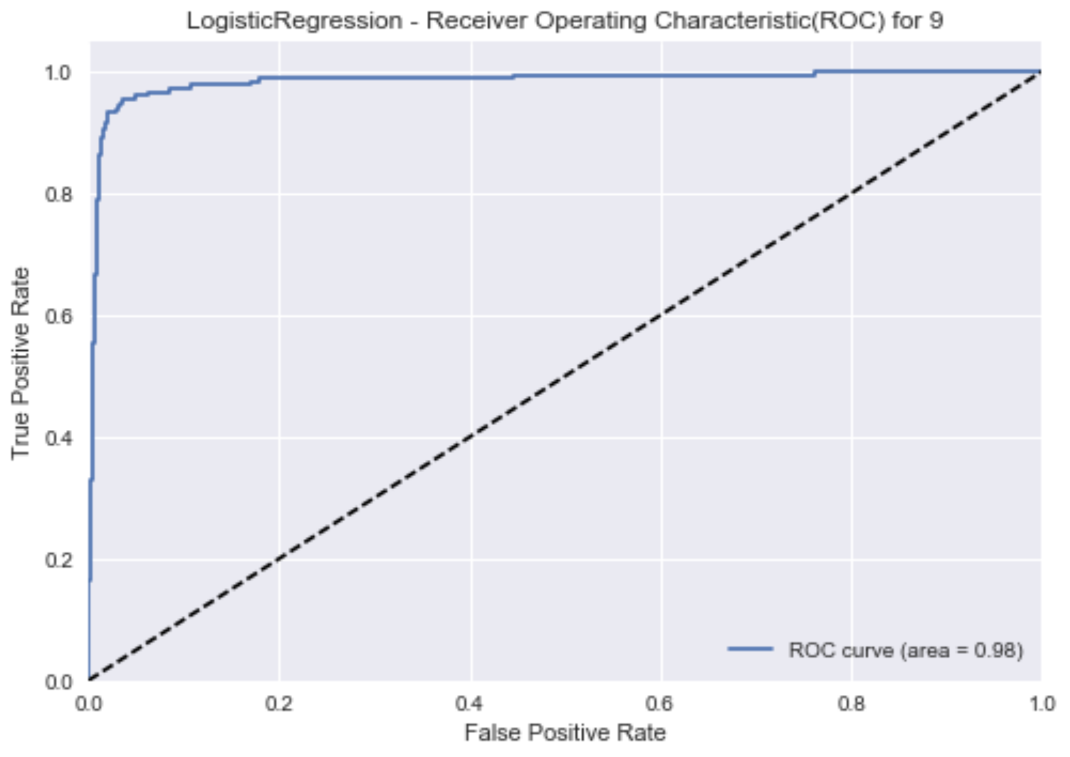

LogisticRegression - Receiver Operating Characteristic(ROC) for 6
1.0
0.8


True Positive F
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
ROC curve (area = 1.00)


LogisticRegression - Receiver Operating Characteristic(ROC) for 7
True Positive Rate
False Positive Rate
ROC curve (area = 0.99)


LogisticRegression - Receiver Operating Characteristic(ROC) for 8
True Positive Rate
ROC curve (area = 0.97)



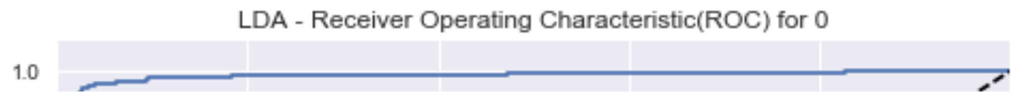


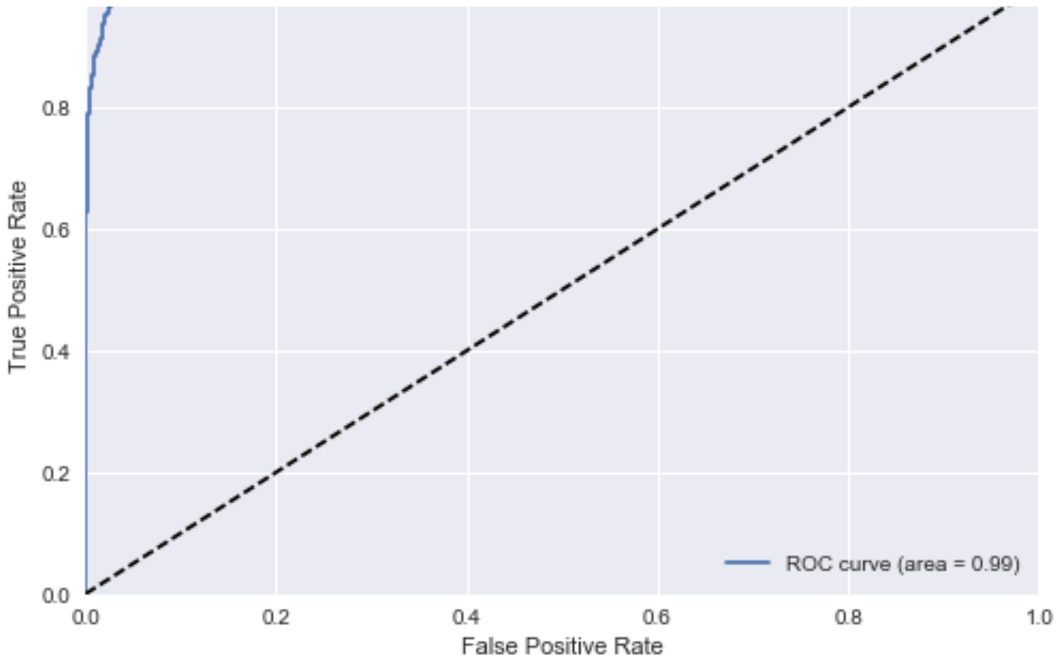
```
In [19]: LDA = OneVsRestClassifier(LinearDiscriminantAnalysis())
```

```
In [20]: y_score = LDA.fit(train_x, train_y).decision_function(test_x)
```

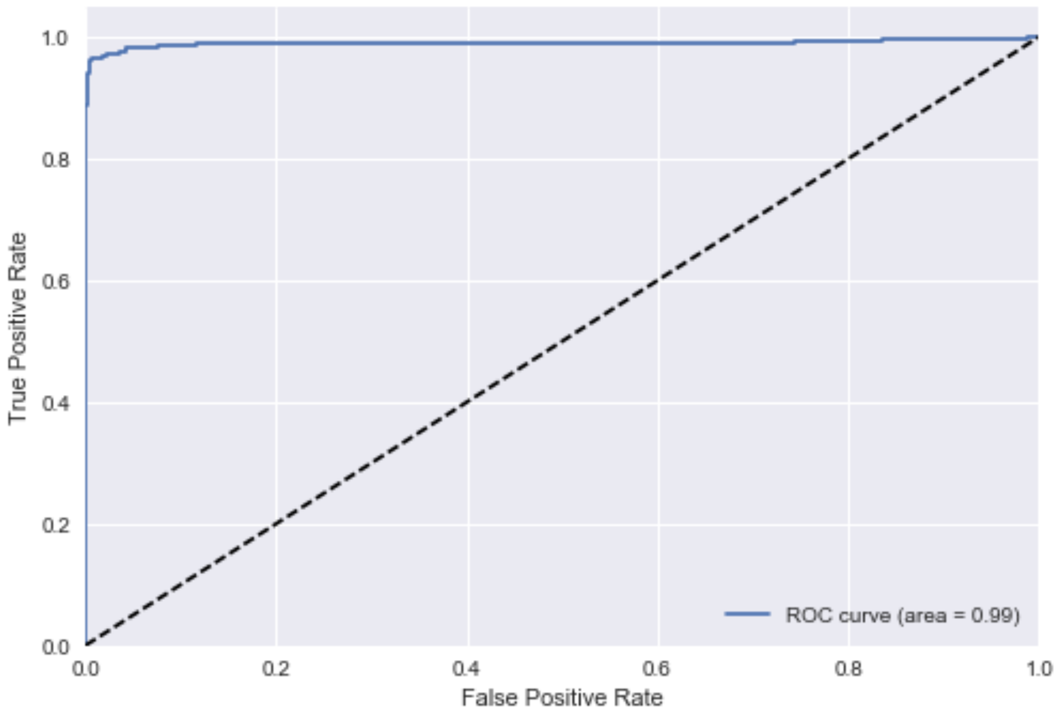
The ROC curves show that LDA peforms worst for the digit 8 giving an area of 0.95.

```
In [21]: fpr = dict()
         tpr = dict()
         roc_auc = dict()
         for i in range(n_classes):
             fpr[i], tpr[i], _ = metrics.roc_curve(test_y[:, i], y_score[:, i])
             roc_auc[i] = metrics.auc(fpr[i], tpr[i])

         for i in range(n_classes):
             plt.figure()
             plt.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f)' % roc_au
             plt.plot([0, 1], [0, 1], 'k--')
             plt.xlim([0.0, 1.0])
             plt.ylim([0.0, 1.05])
             plt.xlabel('False Positive Rate')
             plt.ylabel('True Positive Rate')
             plt.title('LDA - Receiver Operating Characteristic(ROC) for ' + st
             plt.legend(loc="lower right")
             plt.show()
```

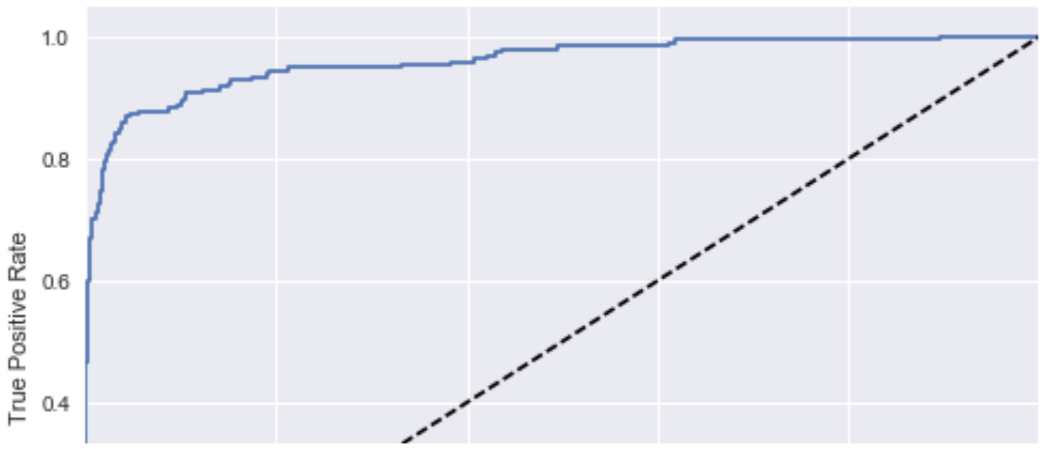


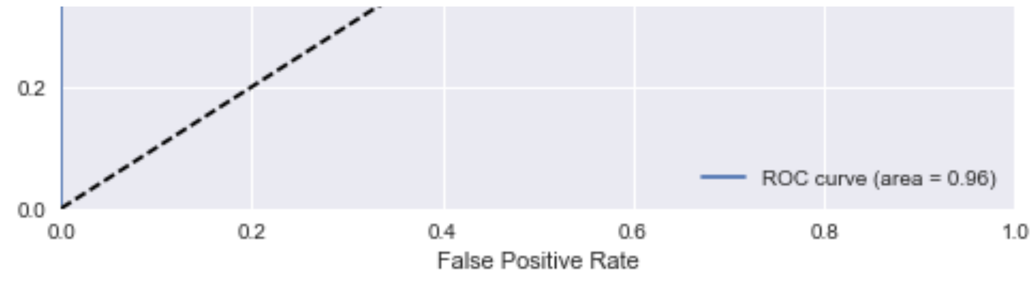


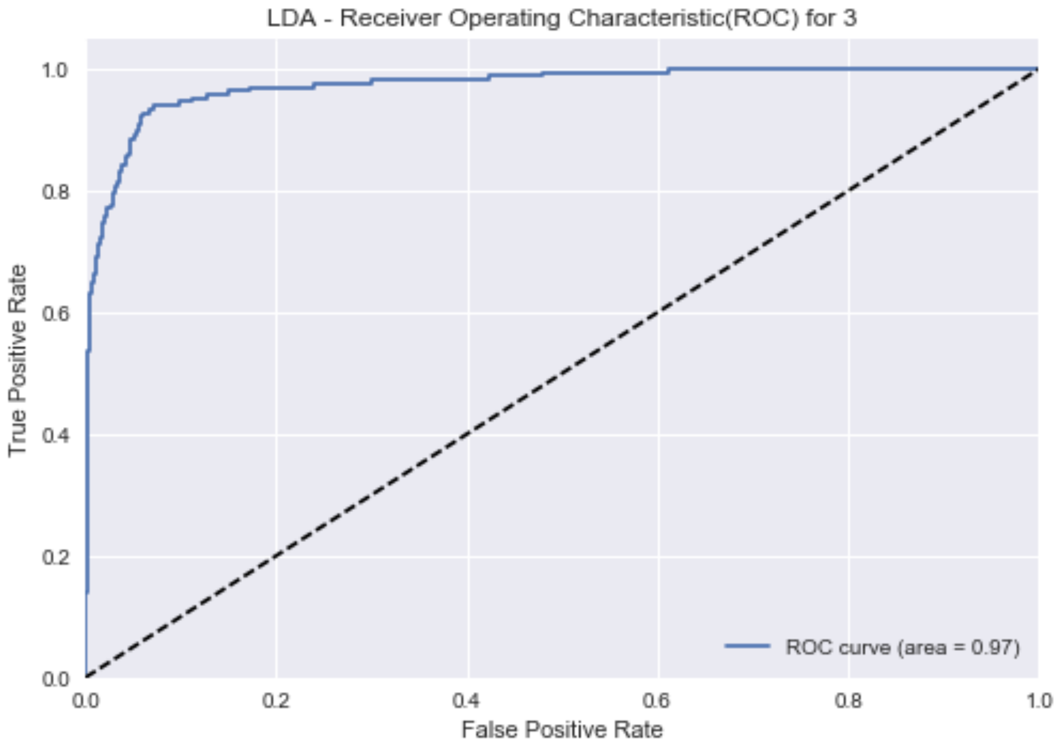
LDA - Receiver Operating Characteristic(ROC) for 1

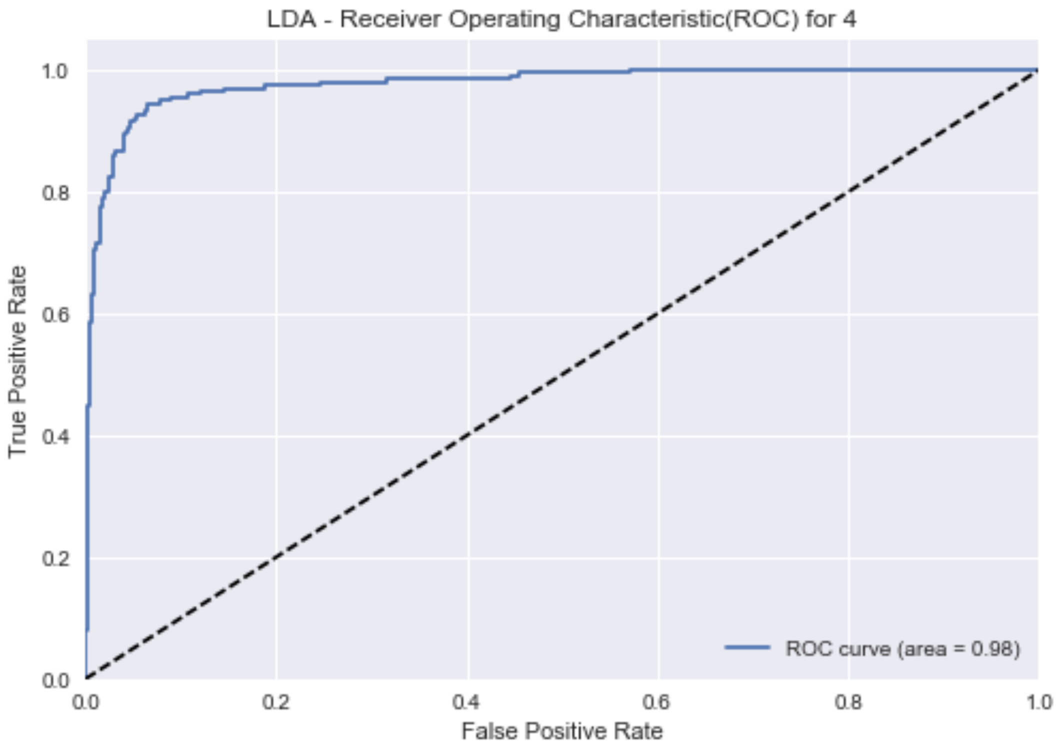


LDA - Receiver Operating Characteristic(ROC) for 2

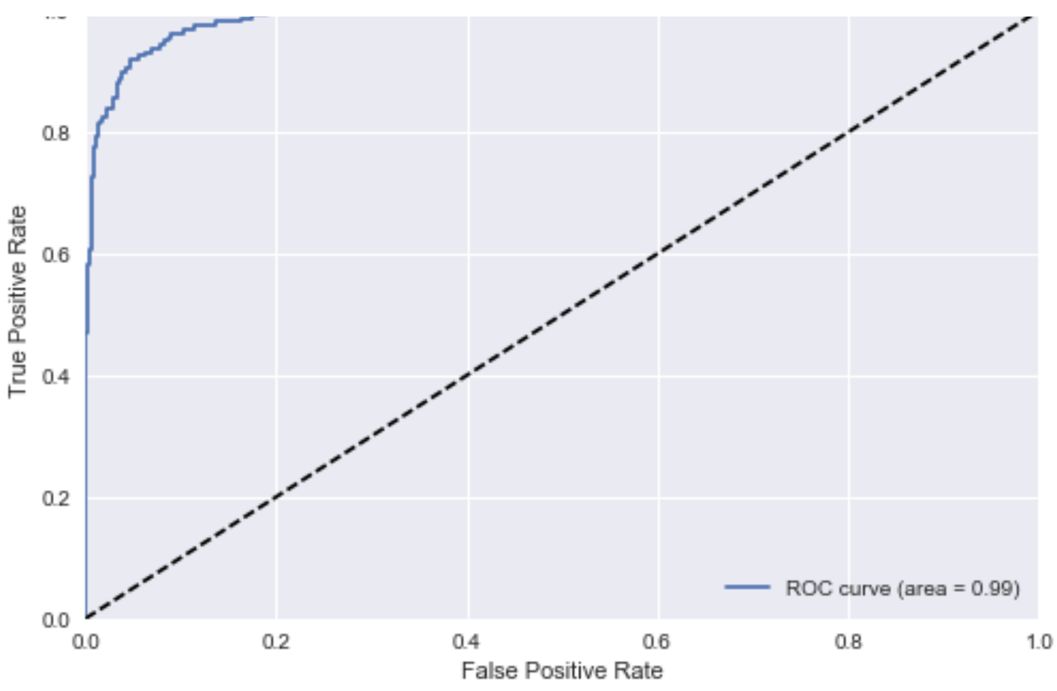



0.2
0.0
ROC curve (area = 0.96)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

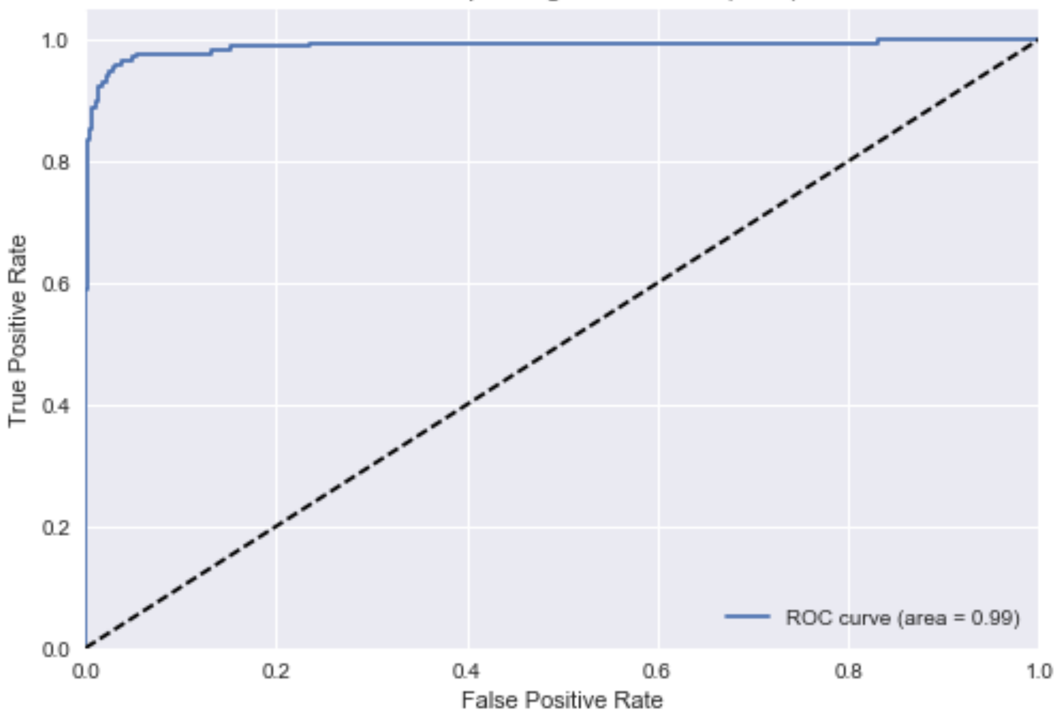

LDA - Receiver Operating Characteristic(ROC) for 3
True Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
ROC curve (area = 0.97)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate


LDA - Receiver Operating Characteristic(ROC) for 4
True Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
ROC curve (area = 0.98)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

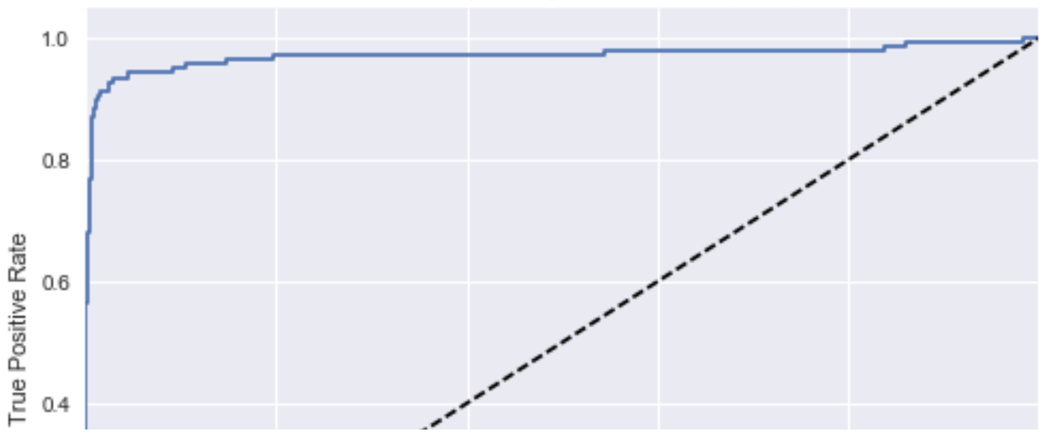
LDA - Receiver Operating Characteristic(ROC) for 5
1.0

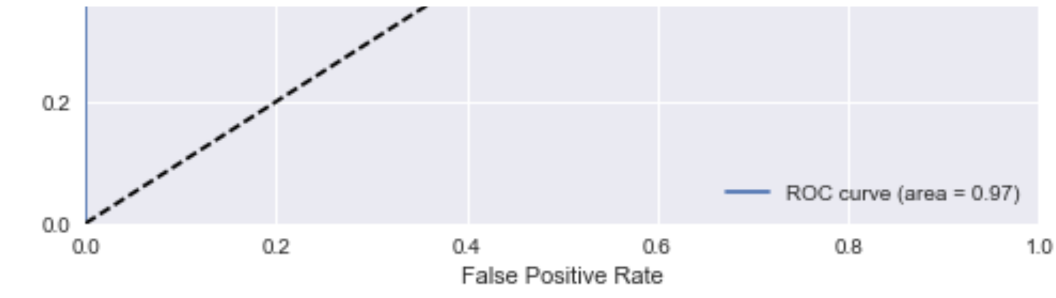


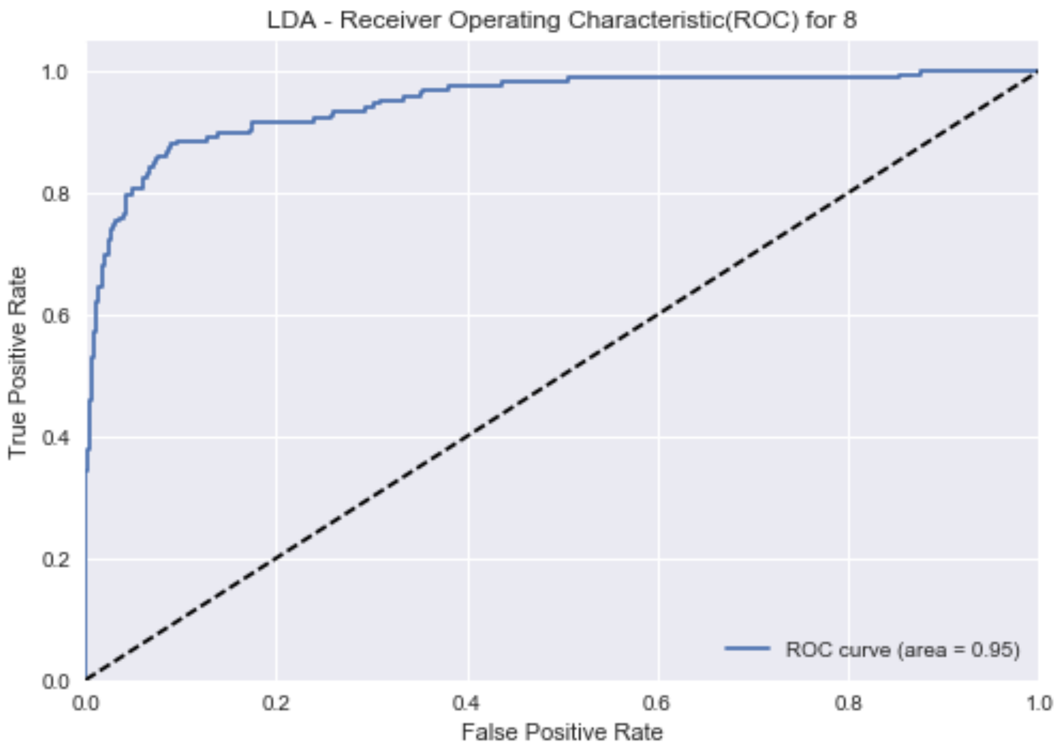
LDA - Receiver Operating Characteristic(ROC) for 6

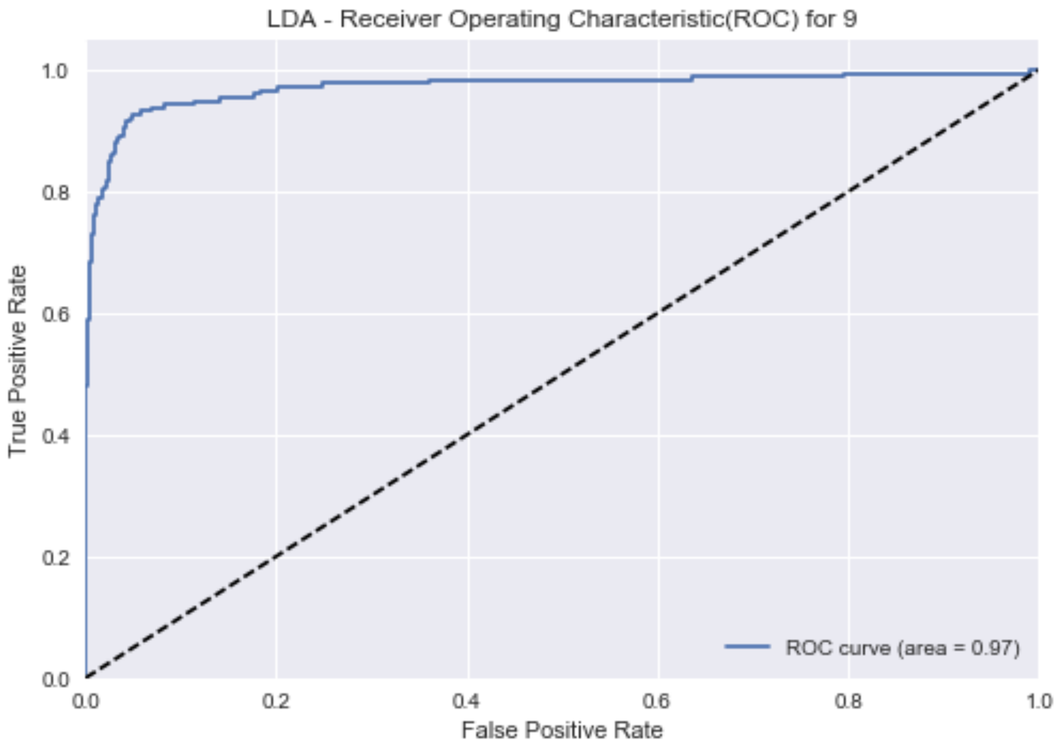


LDA - Receiver Operating Characteristic(ROC) for 7




0.2
0.0
ROC curve (area = 0.97)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

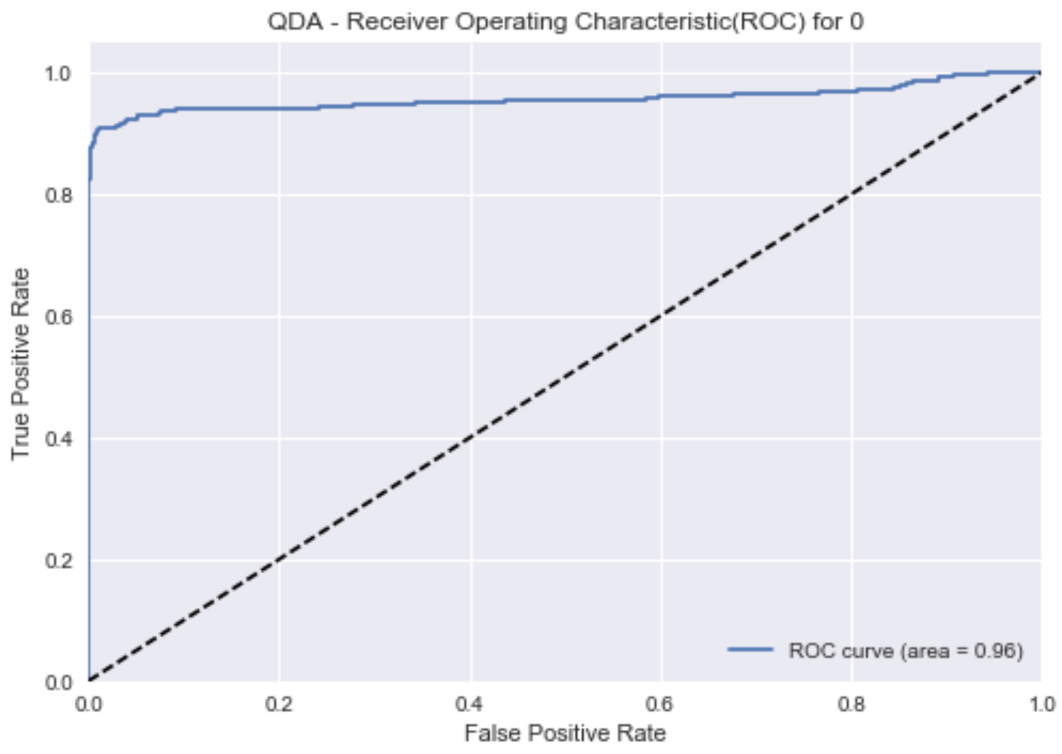

LDA - Receiver Operating Characteristic(ROC) for 8
True Positive Rate
False Positive Rate
ROC curve (area = 0.95)

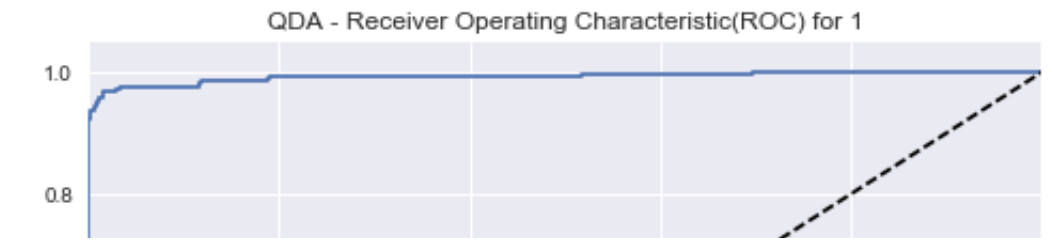

LDA - Receiver Operating Characteristic(ROC) for 9
True Positive Rate
False Positive Rate
ROC curve (area = 0.97)

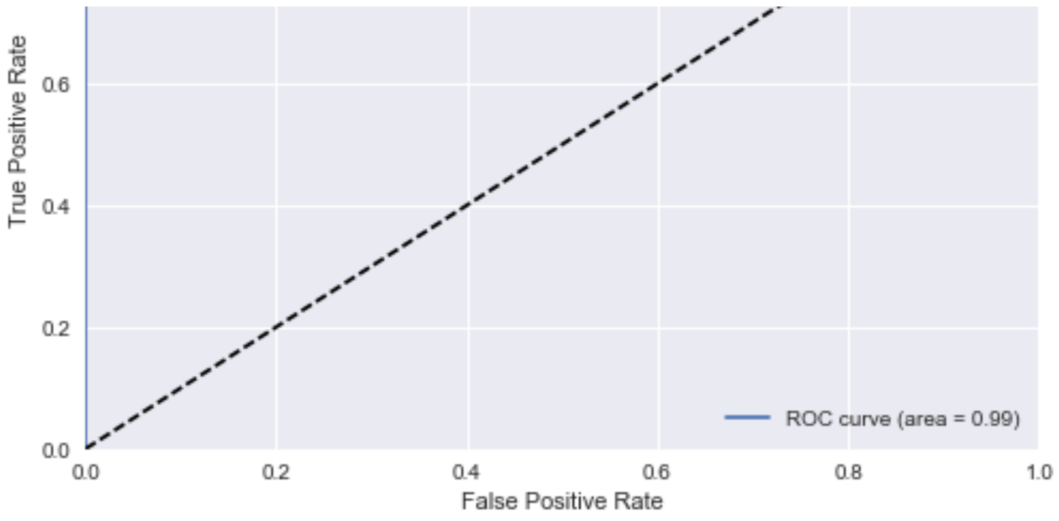
```
In [22]: QDA = OneVsRestClassifier(QuadraticDiscriminantAnalysis())
         y_score = QDA.fit(train_x, train_y).decision_function(test_x)
```

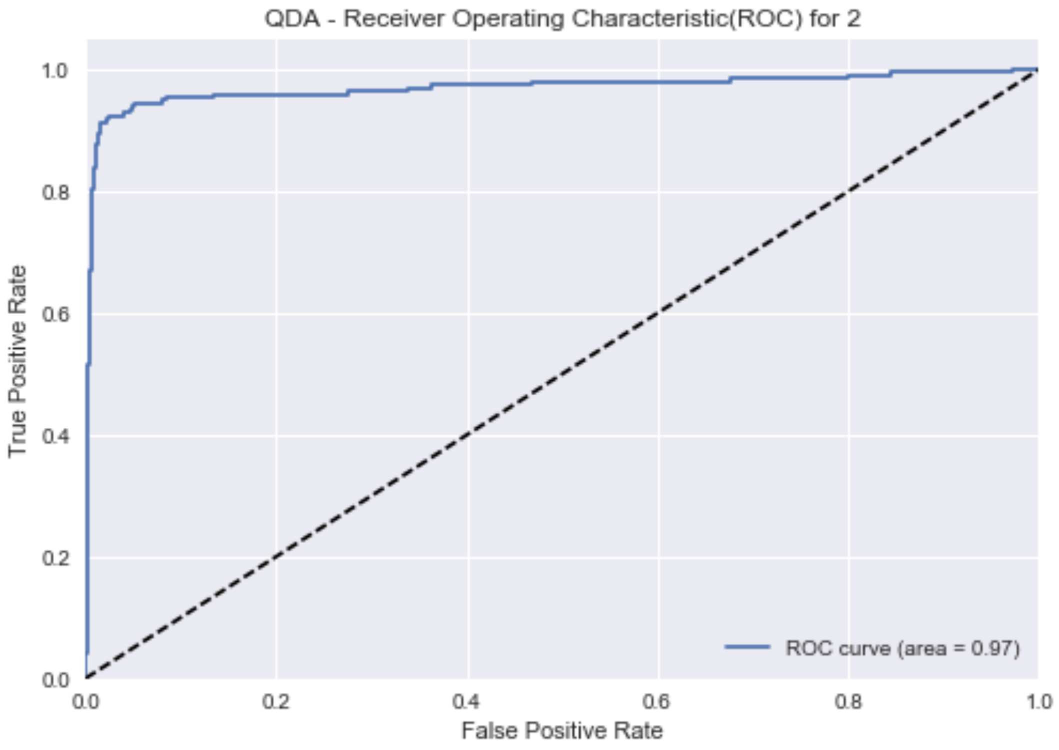
The ROC curves show that QDA peforms worst for the digit 8 giving an area of 0.79.

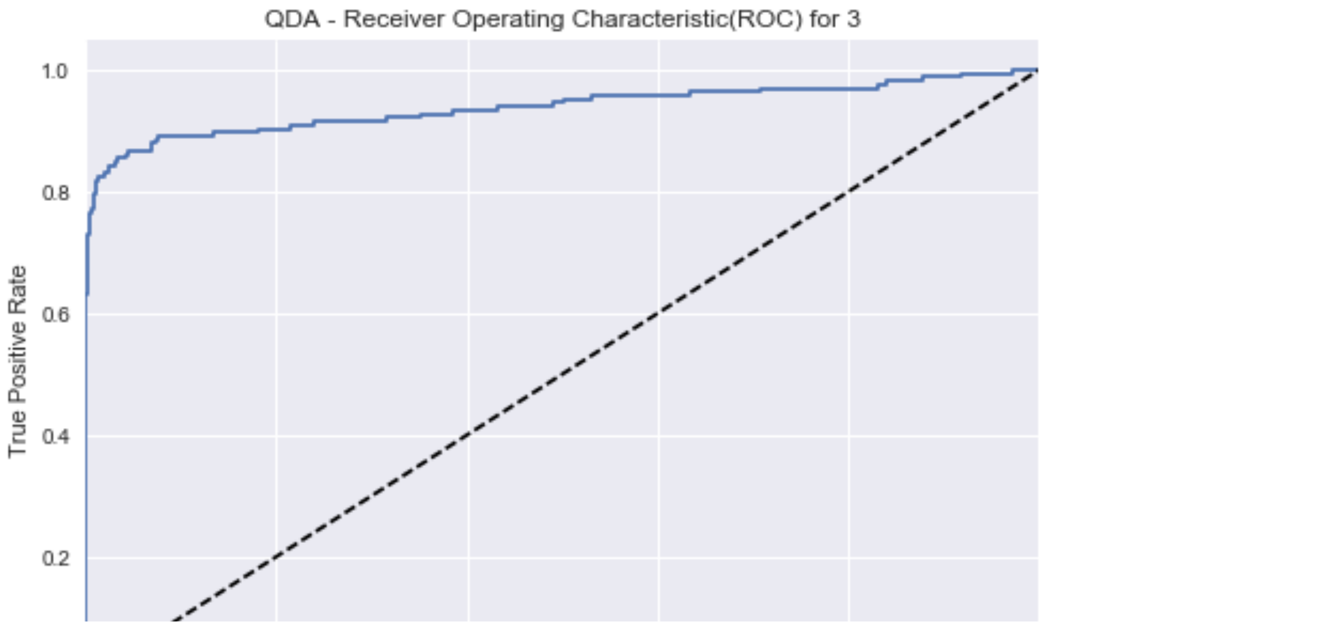
```
In [23]: fpr = dict()
         tpr = dict()
         roc_auc = dict()
         for i in range(n_classes):
             fpr[i], tpr[i], _ = metrics.roc_curve(test_y[:, i], y_score[:, i])
             roc_auc[i] = metrics.auc(fpr[i], tpr[i])

         for i in range(n_classes):
             plt.figure()
             plt.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f)' % roc_au
             plt.plot([0, 1], [0, 1], 'k--')
             plt.xlim([0.0, 1.0])
             plt.ylim([0.0, 1.05])
             plt.xlabel('False Positive Rate')
             plt.ylabel('True Positive Rate')
             plt.title('QDA - Receiver Operating Characteristic(ROC) for ' + st
             plt.legend(loc="lower right")
             plt.show()
```



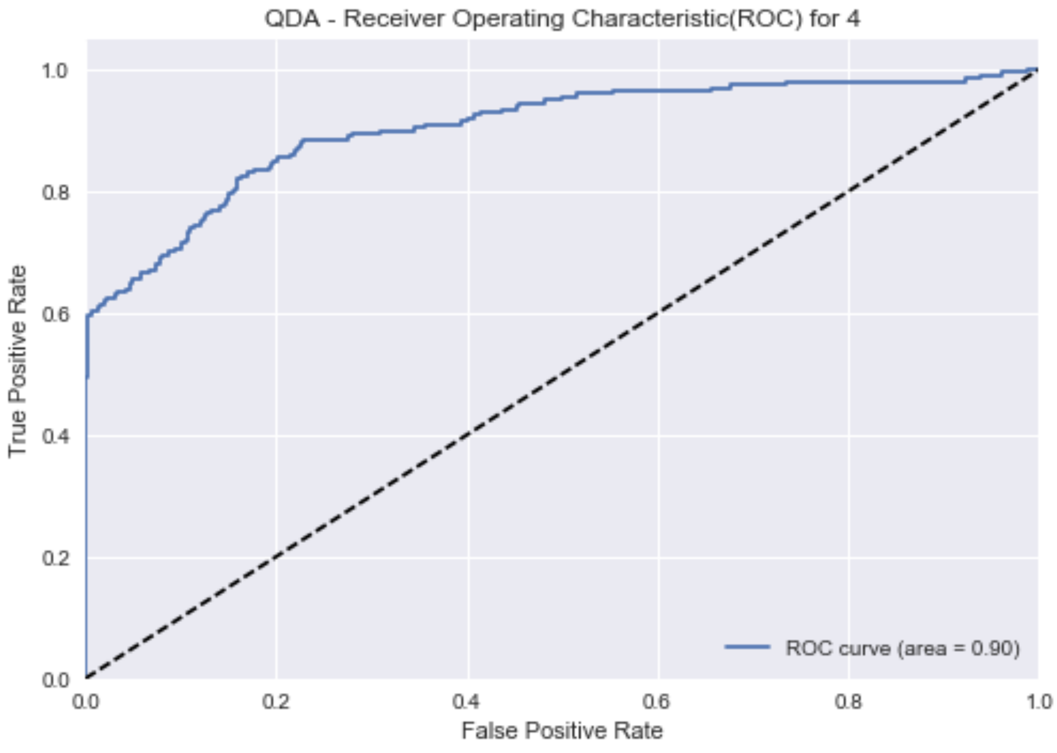


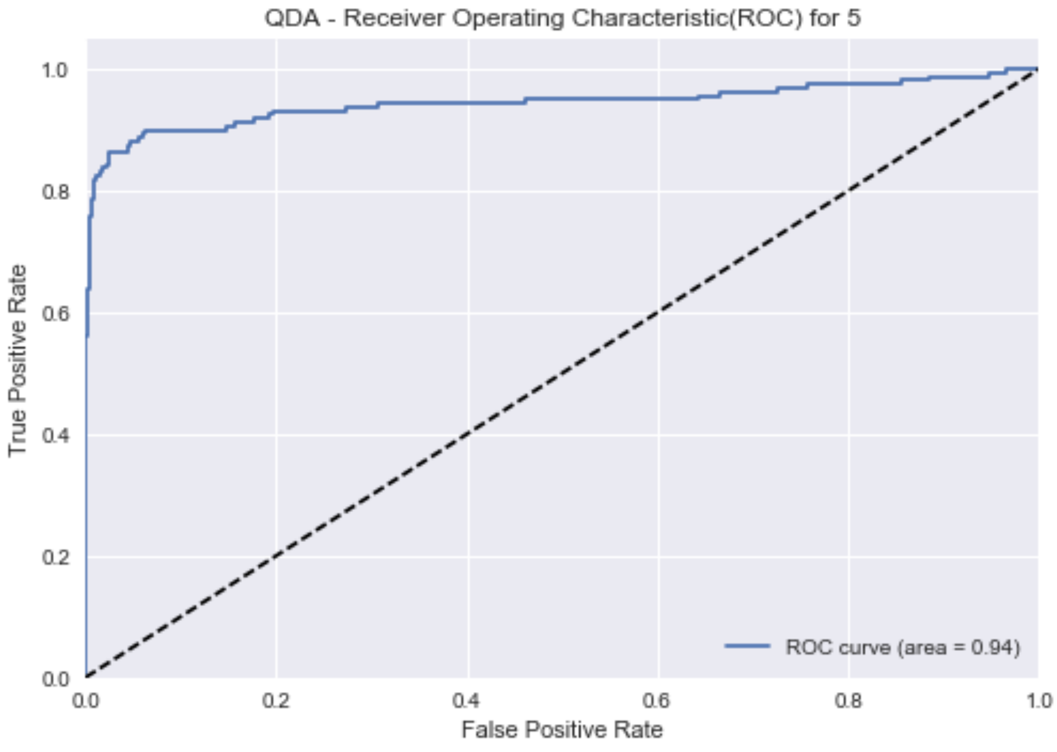

True Positive Rate
0.0
0.2
0.4
0.6
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
ROC curve (area = 0.99)

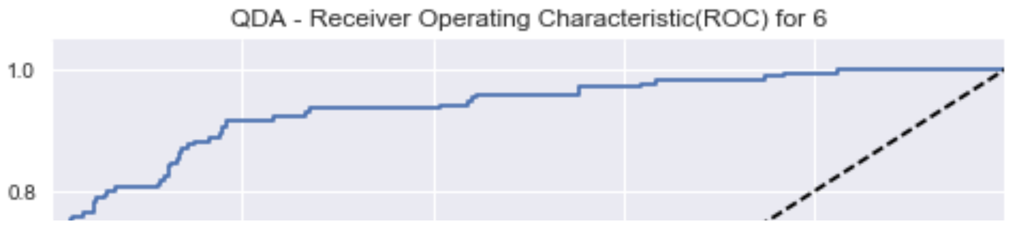

QDA - Receiver Operating Characteristic(ROC) for 2
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
ROC curve (area = 0.97)

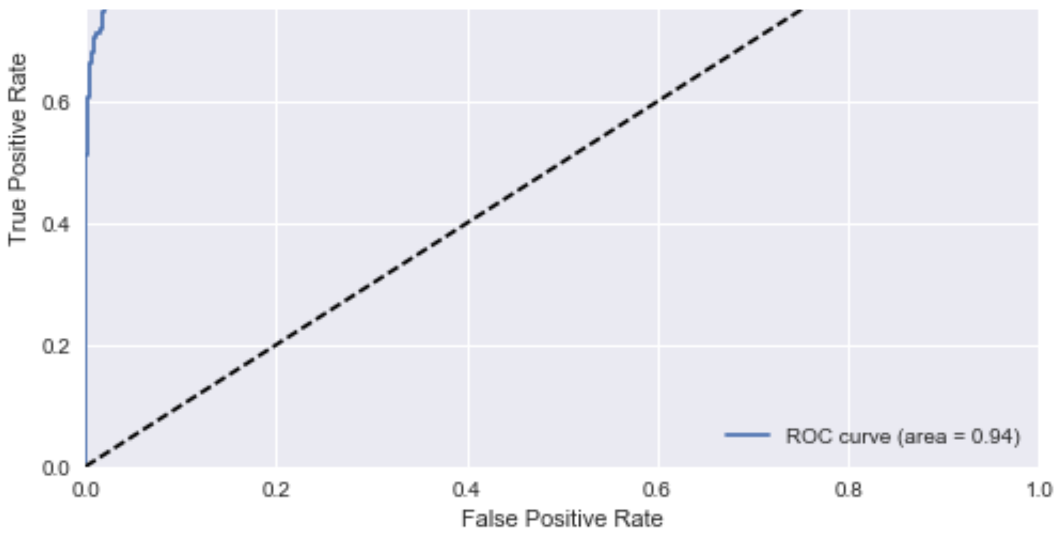

QDA - Receiver Operating Characteristic(ROC) for 3
True Positive Rate
0.2
0.4
0.6
0.8
1.0

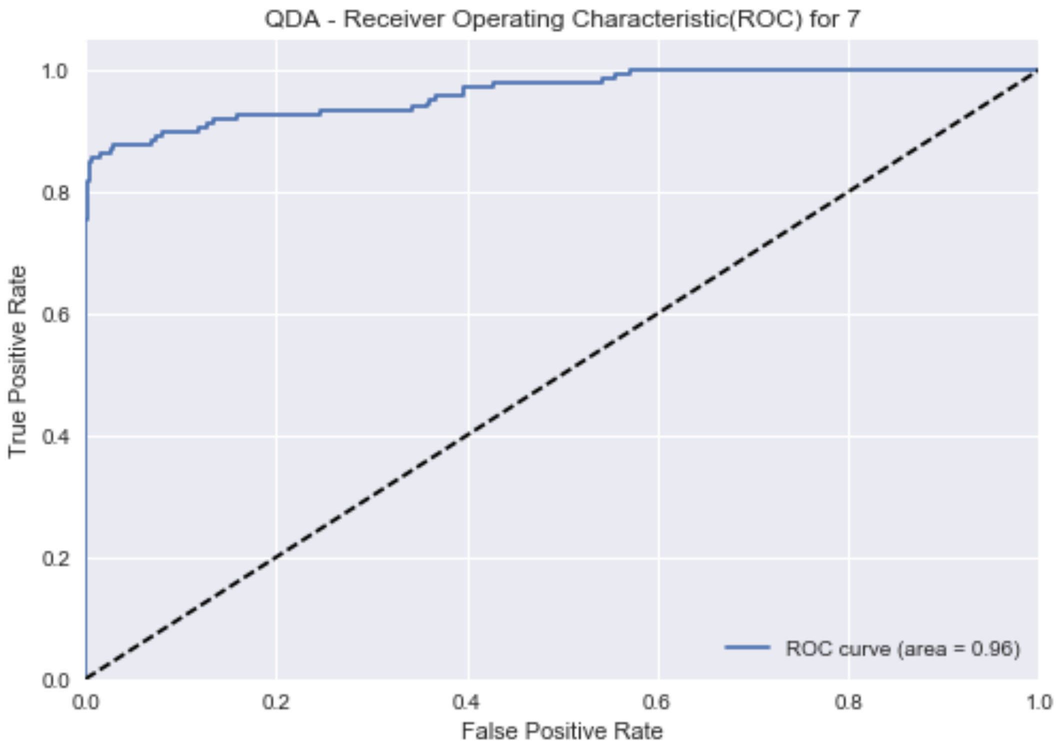
0.0
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
ROC curve (area = 0.94)

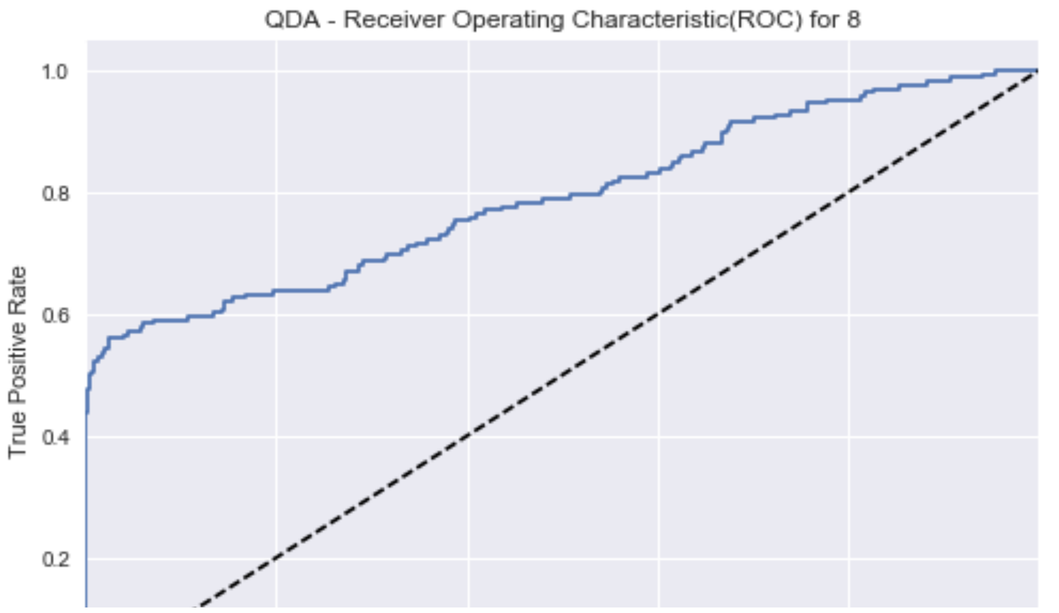

QDA - Receiver Operating Characteristic(ROC) for 4
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.90)

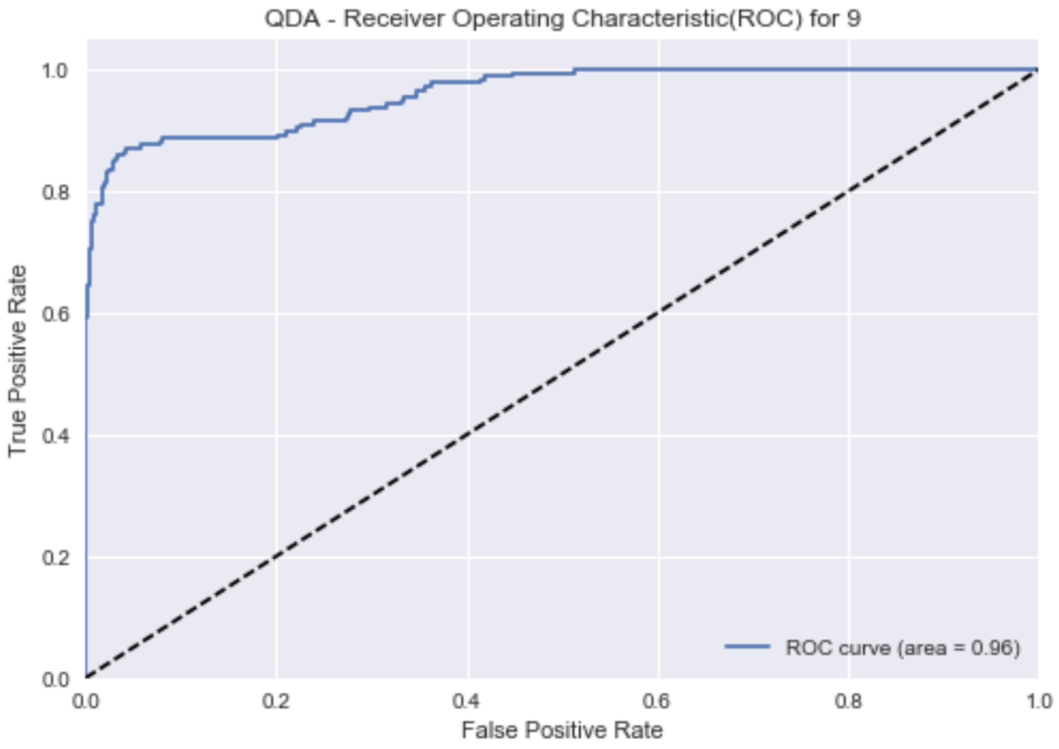

QDA - Receiver Operating Characteristic(ROC) for 5
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.94)


QDA - Receiver Operating Characteristic(ROC) for 6
1.0
0.8


True Positive Rate
0.0
0.2
0.4
0.6
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.94)


QDA - Receiver Operating Characteristic(ROC) for 7
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.96)


QDA - Receiver Operating Characteristic(ROC) for 8
True Positive Rate
0.2
0.4
0.6
0.8
1.0



Based on tha above analysis, it seems that LogisticRegression has the best accuracy and ROC curves (on average for all digits).