Week #8 assignment - CS156

October 29, 2017

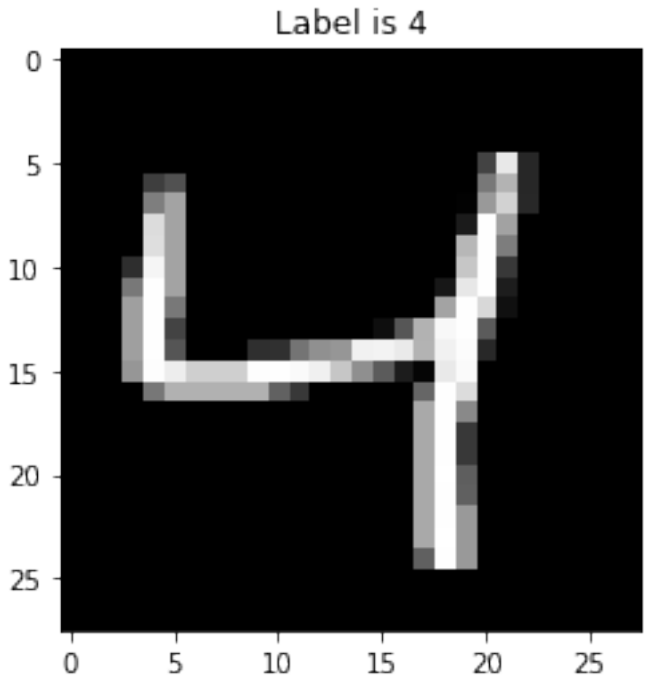
1 Ricardo Montalvo Guzman

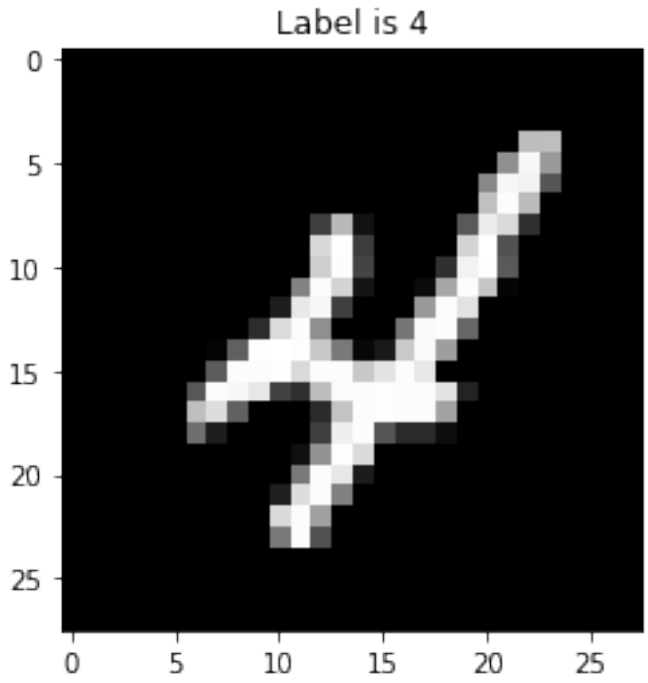
I used a support vector classifier with three kernels (linear, poly (degree = 3) and rbf) to classify 4's and 9's from the MNIST dataset. Examples of these written digits are plotted in the first cell below this.

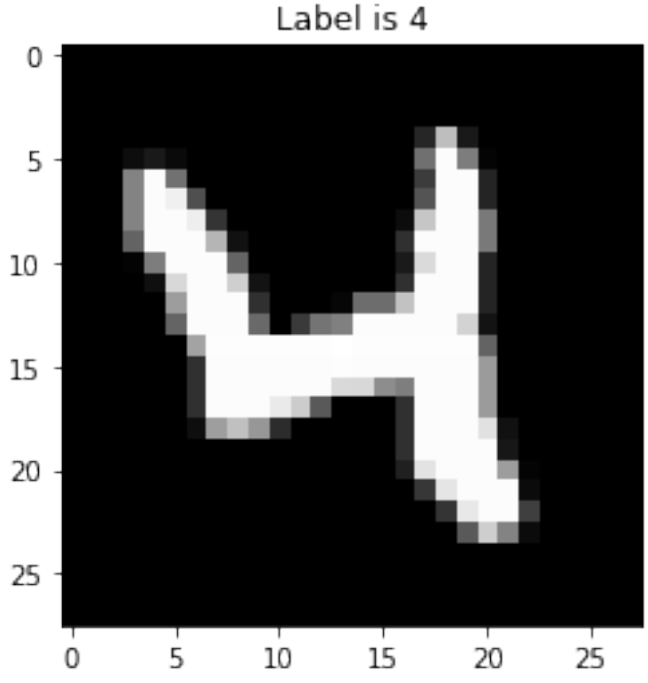
As I encountered issues with the running time, I reduced my datasets using LDA. This was fitted to the training set (comprising 60% of observations) and subsequently used to transform the test set. The kernel with the best predictions for the test set with reduced dimensions (accuracy score) was the linear one, followed by poly and then rbf (.962, .961 and .96, respectively). As for time, they took .29, .20, .39 s, linear, poly and rbf, respectively.

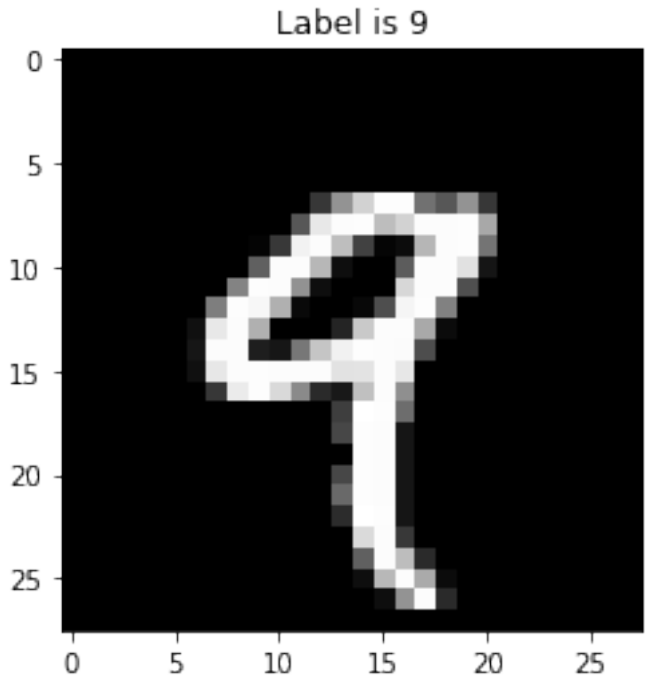
```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.datasets import fetch_mldata

        mnist = fetch_mldata('MNIST original')


        # Fetching the indexes of the classes that I chose,
        # four and nine
        four_index = []
        nine_index = []

        for i in range(len(mnist.target)):
            if mnist.target[i] == 4:
                four_index.append(i)
            elif mnist.target[i] == 9:
                nine_index.append(i)

        # Getting rid of the instances outside of the classes I selected
        # using the lists of indexes generated previously
        # for the purpose of speeding up running time
        four_data = mnist.data[four_index]
        four_target = mnist.target[four_index]

        nine_data = mnist.data[nine_index]
        nine_target = mnist.target[nine_index]
```

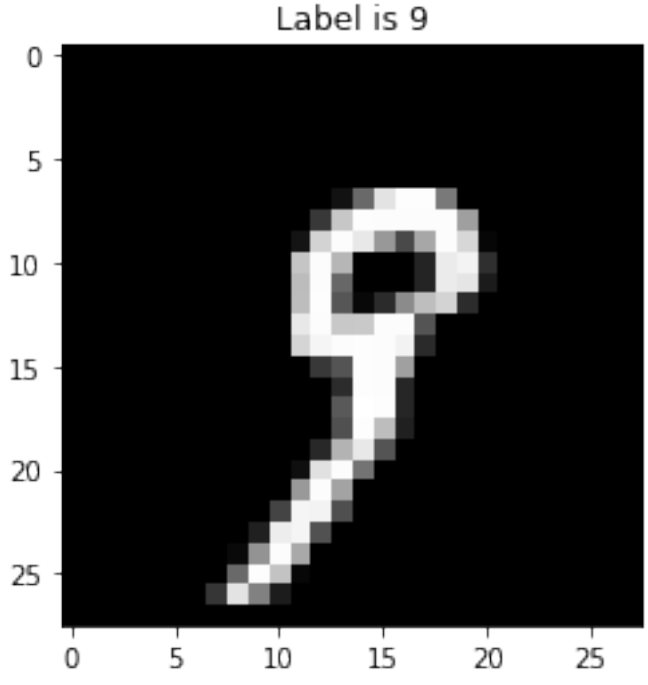
```
# Plotting the three first written digits of each
#four_data and nine_data
for i in range(3):
    pixels = four_data[i]
    pixels = pixels.reshape((28, 28))
    plt.title('Label is 4')
    plt.imshow(pixels, cmap='gray')
    plt.show()

for i in range(3):
    pixels = nine_data[i]
    pixels = pixels.reshape((28, 28))
    plt.title('Label is 9')
    plt.imshow(pixels, cmap='gray')
    plt.show()
```

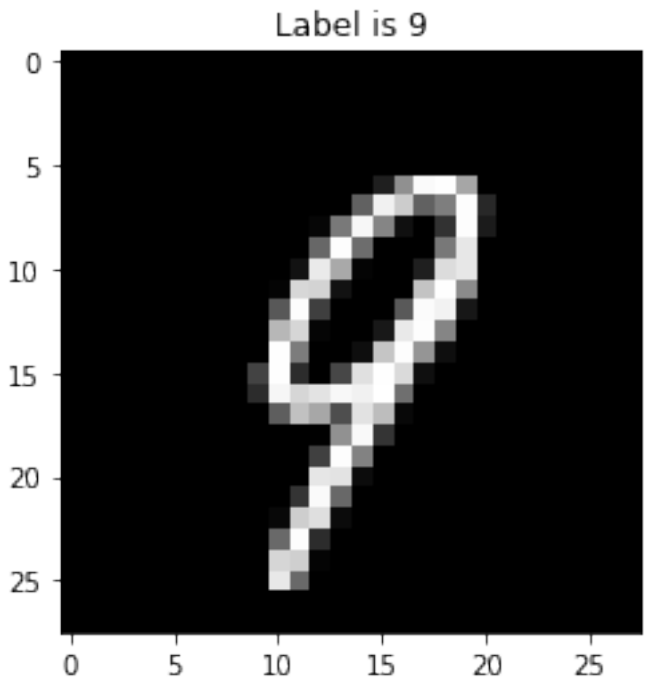



Label is 4
0
5
10
15
20
25
0
5
10
15
20
25


Label is 4
0
5
10
15
20
25
0
5
10
15
20
25


Label is 9
0
5
10
15
20
25
0
5
10
15
20
25


Label is 9
0
5
10
15
20
25
0
5
10
15
20
25



In [2]:
```
import time
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score


# The data and target sets of the fours and nines are reduced to a single one
# for data and target
data = np.concatenate((four_data, nine_data))
target = np.concatenate((four_target, nine_target))

# Now, for the purpose of classifying, data and target are divided using
# train_test_split. The chosen size of the test set is 30% of the original
# dataset
X_train, X_test, y_train, y_test = train_test_split(
data, target, test_size=0.4, random_state=42)

# X_train and X_test are transformed using LDA fitted to X_train, as it is
# commonplace not to let the test data weight in the processing of the data
lda = LinearDiscriminantAnalysis(n_components = 2)
lda_X_train = lda.fit(X_train, y_train).transform(X_train)
lda_X_test = lda.fit(X_train, y_train).transform(X_test)
```

```
        # LINEAR SVM TRAINING
        clf_linear = SVC(kernel = 'linear')

        linear_start = time.clock()
        clf_linear.fit(lda_X_train, y_train)
        print "Training time of SVC with linear kernel is", time.clock() - linear_start

        y_predictions_test = clf_linear.predict(lda_X_test)
        y_predictions_train = clf_linear.predict(lda_X_train)

        print "Accuracy score of SVC with linear kernel on training set", accuracy_score(y_tra

        print "Accuracy score of SVC with linear kernel on test set", accuracy_score(y_test, y_
```

```
/Users/ricardomontalvoguzman/anaconda/lib/python2.7/site-packages/sklearn/discriminant_analysis
  warnings.warn("Variables are collinear.")
```

```
Training time of SVC with linear kernel is 0.291384
Accuracy score of SVC with linear kernel on training set 0.9691619301
Accuracy score of SVC with linear kernel on test set 0.961908216942
```

```
In [3]: # POLY SVM TRAINING
        clf_poly = SVC(kernel = 'poly', degree = 3)

        poly_start = time.clock()
        clf_poly.fit(lda_X_train, y_train)
        print "Training time of SVC with poly kernel is", time.clock() - poly_start

        y_predictions_test = clf_poly.predict(lda_X_test)
        y_predictions_train = clf_poly.predict(lda_X_train)

        print "Accuracy score of SVC with poly kernel on train set", accuracy_score(y_train, y_

        print "Accuracy score of SVC with poly kernel on test set", accuracy_score(y_test, y_pr
```

```
Training time of SVC with poly kernel is 0.203753
Accuracy score of SVC with poly kernel on train set 0.969040996493
Accuracy score of SVC with poly kernel on test set 0.96063849084
```

```
In [4]: # RBF SVM TRAINING
        clf_rbf = SVC(kernel = 'rbf')

        rbf_start = time.clock()
        clf_rbf.fit(lda_X_train, y_train)
        print "Training time of SVC with rbf kernel is", time.clock() - rbf_start
```

```
        y_predictions_test = clf_rbf.predict(lda_X_test)
        y_predictions_train = clf_rbf.predict(lda_X_train)

        print "Accuracy score of SVC with rbf kernel on train set", accuracy_score(y_train, y_p

        print "Accuracy score of SVC with rbf kernel on test set", accuracy_score(y_test, y_pre
```

```
Training time of SVC with rbf kernel is 0.390654
Accuracy score of SVC with rbf kernel on train set 0.970734066997
Accuracy score of SVC with rbf kernel on test set 0.959912933067
```