Hong Seh Looi
frank@minerva.kgi.edu

Assignment 4
CS156: Machine Learning for Science and Profit

1. **Load the entire MNIST digit dataset**

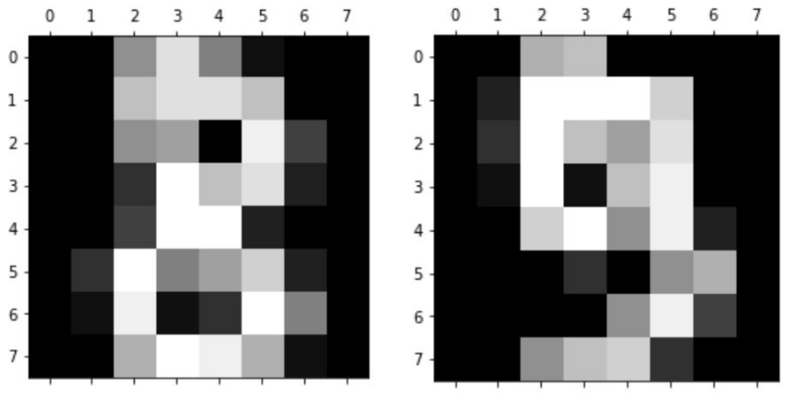
```
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn import svm
from sklearn.metrics import classification_report, accuracy_score
import time
import numpy as np
from sklearn.datasets import load_digits

# MNIST dataset in CSV obtained from https://pjreddie.com/projects/mnist-in-csv/
# Same size and content as in http://yann.lecun.com/exdb/mnist/
# Column 1 is label and Columns 2 - 785 are pixel data of each image
# Open train and test dataset as Panda DataFrame
train_data = pd.read_csv('mnist_train.csv')
test_data = pd.read_csv('mnist_test.csv')

# Concatenate train and test data
data = pd.concat([train_data,test_data])

# Select classes 8 and 9 from the dataset
data = data.drop(data[data.Label < 8].index)
print "The size of the data with classes 8 and 9 is:",data.shape[0]
```

2. **Choose two digit classes (e.g 7s and 3s) from the training data, and plot some of the examples.**



3. **Train a support vector classifier using each of the following kernels:**
   a. **Linear**
   b. **Poly**
   c. **RBF**
4. **Report your training times on the dataset for the different kernels.**
5. **Report your error rates on the testing dataset for the different kernels.**

Simple implementation

```
# Dropping the label column for the X training and testing sets
# Using the label column for the Y training and testing sets
# Splitting into 80% Training and 20% Testing set
# Larger training set is used to reduce overfitting since SVM is sensitive to training data
X, X_test, Y, Y_test =
train_test_split(data.drop(['Label'],axis=1),data.Label,test_size=.2,random_state=42)

# Declaring 3 different kernals for Support Vector Classifier
kernels = ['linear','poly','rbf']

# Fit models using each kernel and computing performance of each model
for kernel in kernels:
    print "Using a {} kernel".format(kernel)

    start_time = time.time()
    clf = svm.SVC(kernel=kernel, random_state = 35)
    clf.fit(X, Y)
    predictions = clf.predict(X_test)

    print "The accuracy of this kernel is %.2f" % accuracy_score(Y_test, predictions)
    print classification_report(Y_test, predictions)
    print "Time taken: %.2fs" % (time.time() - start_time)
    print ""

# NOTE on using svm.SVC:
# Limited flexibility on choice of penalties and loss functions to reduce overfitting
# Implementation based on libsvm rather than liblinear
# Time complexity is more than O(n^2)
```

```
Using a linear kernel
The accuracy of this kernel is 0.97
             precision    recall  f1-score   support

          8       0.97      0.97      0.97      1002
          9       0.97      0.97      0.97       968

avg / total       0.97      0.97      0.97      1970

Time taken: 6.23s

Using a poly kernel
The accuracy of this kernel is 1.00
             precision    recall  f1-score   support

          8       1.00      0.99      1.00      1002
          9       0.99      1.00      1.00       968

avg / total       1.00      1.00      1.00      1970

Time taken: 5.73s

Using a rbf kernel
The accuracy of this kernel is 0.49
             precision    recall  f1-score   support

          8       0.00      0.00      0.00      1002
          9       0.49      1.00      0.66       968

avg / total       0.24      0.49      0.32      1970

Time taken: 101.40s
```

Advanced Implementation

- Finding the best penalty term, C, for each model since choice of penalties is important in the C-Support Vector Classification
- Using cross validation to measure each model's performance
- Goal: Reduce overfitting and improve generalizability

```
# MNIST dataset in CSV obtained from https://pjreddie.com/projects/mnist-in-csv/
# Same size and content as in http://yann.lecun.com/exdb/mnist/
# Column 1 is label and Columns 2 - 785 are pixel data of each image
# Open train and test dataset as Panda DataFrame
train_data = pd.read_csv('mnist_train.csv')
test_data = pd.read_csv('mnist_test.csv')

# Concatenate train and test data
data = pd.concat([train_data,test_data])

# Select classes 8 and 9 from the dataset
data = data.drop(data[data.Label < 8].index)
print "The size of the data with classes 8 and 9 is:",data.shape[0]

# Dropping the label column for the X training and testing sets
# Using the label column for the Y training and testing sets
# Splitting into 80% Training and 20% Testing set
# Larger training set is used to reduce overfitting since SVM is sensitive to training data
X, X_test, Y, Y_test =
train_test_split(data.drop(['Label'],axis=1),data.Label,test_size=.2,random_state=42)

# Further splitting training set into validation set to find optimal value of c
X_train, X_val, Y_train, Y_val = train_test_split(X, Y,test_size=0.2, random_state=84)

# Declaring 3 different kernals for Support Vector Classifier
kernels = ['linear','poly','rbf']

# Test best values of C, the penalty term, for each kernel model
# C-values are floats between 0 and 1, step size is 0.05
cVals = [round(0.05*m,2) for m in range(0*20+1, 1*20+1)] # Returns [0.05, 0.1...1.0]

# Fit models using each kernel and computing performance of each model
for kernel in kernels:
    print ""
    print "Using a {} kernel".format(kernel)
    # For each kernel, find the C-value that produces the highest accuracy on validation
data
    # Array to store accuracy score of each C-value
    accuracies = []
    print("Testing the best value of c on validation dataset of size %d" % len(X_val))
    for c in cVals:
        model = svm.SVC(C=c, kernel=kernel, random_state = 35)
        model.fit(X_val, Y_val)
        score = model.score(X_val, Y_val)
        accuracies.append(score)
    index = np.argmax(accuracies) # index of the cVal with highest accuracy
    print ("The best value of c for the %s kernel is %.2f with accuracy score of %.2f on the
validation data"
           % (kernel, cVals[index], accuracies[index] * 100))

    # Use the most accurate C-value to implement the model on training data
    start_time = time.time()
    clf = svm.SVC(C=cVals[index],kernel=kernel, random_state = 35)
    clf.fit(X_train, Y_train)
    predictions = clf.predict(X_test)

    # Cross validate 5 times and obtain the average accuracy score
    # Cross validation used to improve generalizability of the model
    average_score = np.mean(cross_val_score(clf, X_test, Y_test, scoring = 'accuracy',
cv=5))
    print "The accuracy of this kernel is %.2f" % average_score
    print classification_report(Y_test, predictions)
    print "Time taken: %.2fs" % (time.time() - start_time)

# NOTE on using svm.SVC:
# Limited flexibility on choice of penalties and loss functions to reduce overfitting
# Implementation based on libsvm rather than liblinear
# Time complexity is more than O(n^2)
```

```
Using a linear kernel
Testing the best value of c on validation dataset of size 1576
The best value of c for the linear kernel is 0.05 with accuracy score of 100.00 on the validation data
The accuracy of this kernel is 0.97
             precision    recall  f1-score   support

          8       0.97      0.97      0.97      1002
          9       0.97      0.97      0.97       968

avg / total       0.97      0.97      0.97      1970

Time taken: 5.82s

Using a poly kernel
Testing the best value of c on validation dataset of size 1576
The best value of c for the poly kernel is 0.05 with accuracy score of 100.00 on the validation data
The accuracy of this kernel is 0.99
             precision    recall  f1-score   support

          8       1.00      0.99      0.99      1002
          9       0.99      1.00      0.99       968

avg / total       0.99      0.99      0.99      1970

Time taken: 6.65s

Using a rbf kernel
Testing the best value of c on validation dataset of size 1576
The best value of c for the rbf kernel is 0.55 with accuracy score of 100.00 on the validation data
The accuracy of this kernel is 0.51
             precision    recall  f1-score   support

          8       0.00      0.00      0.00      1002
          9       0.49      1.00      0.66       968

avg / total       0.24      0.49      0.32      1970

Time taken: 66.23s
```