Name: Mohit Mohit
BU ID: U33745431
CS 542

Programming Report

Question 1.

Develop code for training and testing an SVM classifier with nonlinear kernel. You are welcome to use either formulation described in the textbook (chapter 7). You cannot use an SVM library to complete this assignment. You can use quadratic programming library if you like. Using your implementation of the SVM classifier, compare multi-class classification performance of two different voting schemes:

i. "one versus the rest"
ii. "one versus one"

Be sure to specify your voting scheme using a method described in the book. To analyze accuracy, you will find it helpful to produce and analyze the multiclass confusion matrix in addition to examining the overall error rate.

Solution

I have implemented the code for SVM in python. It will take care of finding support vectors and margins of the supplied points. I have used polynomial kernel with degree 2 and 6. Also, I have tried using Gaussian kernel with sigma 0.5.

Code Kernel Functions:-

```
@author: mohitbeniwal
"""
from collections import Counter
from scipy.io import loadmat
import numpy as np # linear algebra
from numpy import linalg
import cvxopt
from sklearn.metrics import confusion_matrix

#polynomial kernel with degree 2 by default
def polynomial_kernel(x, y, d=2):
    return (1 + np.dot(x, y)) ** d

# ploynomial kernel with degree 6 by default
def polynomial_kernel_dsix(x, y, d=6):
    return (1 + np.dot(x, y)) ** d

# gaussian kernel with sigma 0.5 by default
def gaussian_kernel(x, y, sigma=0.5):
    return np.exp(-linalg.norm(x-y)**2 / (2 * (sigma ** 2)))
```

In Fit function I have solved the Dual Optimization problem.

$$\arg\max_{\mathbf{a}} \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

$$\text{subject to} \quad a_n \geqslant 0, \quad n = 1, \cdots, N,$$

$$\sum_{n=1}^{N} a_n t_n = 0.$$

To use cvxopt library we need to convert the above equation in a form as below.

Solves the pair of primal and dual convex quadratic programs

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & Gx \preceq h \\ & Ax = b \end{array}$$

and

$$\begin{array}{ll} \text{maximize} & -(1/2)(q + G^T z + A^T y)^T P^\dagger (q + G^T z + A^T y) - h^T z - b^T y \\ \text{subject to} & q + G^T z + A^T y \in \text{range}(P) \\ & z \succeq 0. \end{array}$$

Here,
G: diagonal matrix of of -1s of size m*m

A: label vector of size m*1

q: vector of size m*1
h: vector of zeroes of size m*1

b: scalar with value as 0.

Passing these parameters to the *quadratic programming* interface of cvxopt, we get the solution for x

We then determine the set of support vectors *S* by finding the indices such that $\alpha_m > 0$ and intercept *b* is calculated by:

$$b = T_s - \sum \alpha_m T_m K(X_m, X_s)$$

Code. For fit function:

```
42      def fit(self, K, X, y):
43          m = X.shape[0]
44          # Solving  min 1/2 x^T P x + q^T x
45          G = cvxopt.matrix(-np.eye(m))
46          A = cvxopt.matrix(y, (1,m))
47
48          P = cvxopt.matrix(np.outer(y,y) * K)
49          q = cvxopt.matrix(-np.ones((m,1)))
50          b = cvxopt.matrix(0.0)
51          h = cvxopt.matrix(np.zeros(m))
52
53          cvxopt.solvers.options['show_progress'] = False
54
55          # solve QP problem
56          solution = cvxopt.solvers.qp(P, q, G, h, A, b)
57
58          # Lagrange multipliers
59          alphas = np.ravel(solution['x'])
60          # Support vectors have non zero lagrange multipliers
61          sv = alphas > 1e-5
62          ind = np.arange(len(alphas))[sv]
63          self.a = alphas[sv]
64          self.sv = X[sv]
65          self.sv_y = y[sv]
66          # Intercept
67          self.b = 0
68          for n in range(len(self.a)):
69              self.b += self.sv_y[n]
70              self.b -= np.sum(self.a * self.sv_y * K[ind[n],sv])
71          self.b /= len(self.a)
72          # Weight vector
73          self.w = None
```

In predict function, I classified X_test by using alphas, support vectors and intercept with the values of the fit function. This function takes an argument ‘way’ which defines the voting scheme that can be one_vs_one or one_vs_rest to predict.

```
75      def predict(self, X,scheme_vote):
76          w = np.zeros(len(X)) # wx
77          for i in range(len(X)):
78              summ = 0
79              # alphas, support vectors, sv labels
80              for a, sv_y, sv in zip(self.a, self.sv_y, self.sv):
81                  summ += a * sv_y * self.kernel(X[i], sv)
82              w[i] = summ
83          # 1 represents 'one vs one' and 2 represents 'one vs rest'
84          out =w+self.b
85          if scheme_vote == 1:
86             return np.sign(out)
87          else:
88             return (out)
```

Above functions are present in the SVM_classifier class.

One vs Rest:

To perform One vs Rest we need to build and train out model keeping one out of (0,1…9) class as one and rest which is left as one another class. While predicting one class is defined as +1 and rest classes are defined with label -1
we will take the actual confidence score for each classifier to decide the label by choosing the maximum confidence score.

I have divided the task in 2 parts by making a function to create models and other to predict and plot the accuracy and confusion matrix.

```
def get_one_vs_rest_models(data, kernel_fun):
    predict = [SVM_classifier(kernel_fun) for j in range(10)]
    X_train = data.get('train_samples')
    ovr = SVM_classifier(kernel_fun)
    K = ovr.createKernal(X_train)
    for i in range(10):
        y_train = data.get('train_samples_labels')
        y_train = y_train.ravel()
         #label each digit 1 and  -1
        y_train = 1.0*(y_train==i)-1.0*(y_train!=i)
        ovr = SVM_classifier(kernel_fun)
        ovr.fit(K,X_train, y_train)
        predict[i] = ovr

    return(predict)

def one_vs_rest(data, svm):
    X_test = data.get('test_samples')
    y_test = data.get('test_samples_labels')
    y_test = y_test.ravel()
    y_predict = [[None]* len(y_test) ]*10
    for j in range(10):
        y_predict[j] = svm[j].predict(X_test,2)

    predicted_labels = []*len(y_test)

    for j in range(len(y_test)):
        listC = []
        for i in range(10):
            listC.append(y_predict[i][j])
        predicted_labels.append(np.argmax(listC))


    correct = np.sum(predicted_labels == y_test)
    print(str(correct)+" out of "+str(len(predicted_labels))+ " predictions are correct")
    model_acc = correct/len(predicted_labels)
    conf_mat = confusion_matrix(y_test,predicted_labels)
    print('\nSVM Trained Classifier Accuracy for One VS rest: ', model_acc)
    print('\nConfusion Matrix for One VS rest: \n',conf_mat)
    return(predicted_labels)

```

One vs One:

For this strategy, we will pick classes in pair where both the classes are not same. By doing so we will end up creating 45 models for distinct pair of classes.
We will assign +1 to one class and -1 to the other class, later we will take the count of the which class got the highest count as +1 or -1 and accordingly we will finalize the prediction for the data point.
Same as One vs Rest I have divided the task into 2 parts.

```python
def get_one_vs_one_models(data, kernel_fun):
    models = [[SVM_classifier(kernel_fun) for j in range(10)] for i in range(10)]

    for i in range(10):
        for j in range(10):
            if i != j and i<j:
                X_train = data.get('train_samples')
                y_train = data.get('train_samples_labels')
                y_train = y_train.ravel()
                X_train = X_train[(y_train==i)|(y_train==j)]
                y_train = y_train[(y_train==i)|(y_train==j)]
                #label each digit 1 and  -1
                y_train = 1.0*(y_train==i)-1.0*(y_train==j)

                ovo = SVM_classifier(kernel_fun)
                kernal = ovo.createKernal(X_train)
                ovo.fit(kernal,X_train, y_train)
                models[i][j] = ovo

    return(models)


def one_vs_one(data, svm_models):
    X_test = data.get('test_samples')
    y_test = data.get('test_samples_labels')
    y_pred = [[[None] for j in range(10)] for i in range(10)]
    y_test = y_test.ravel()

    for i in range(10):
        for j in range(10):
            if i != j and i<j:
                preds = svm_models[i][j].predict(X_test,1)
                pred_label = [i if x == 1 else j for x in preds]
                y_pred[i][j] = pred_label


    predicted_labels = []*len(y_test)

    for k in range(len(y_test)):
        label_list = []
        for i in range(10):
            for j in range(10):
                if i != j and i<j:
                    label_list.append(y_pred[i][j][k])
         # Returns the highest occurring item
        label = Counter(label_list).most_common(1)[0][0]
        predicted_labels.append(label)

    correct = np.sum(predicted_labels == y_test)
    print(str(correct)+" out of "+str(len(predicted_labels))+ " predictions are correct")
    model_acc = correct/len(predicted_labels)
    conf_mat = confusion_matrix(y_test,predicted_labels)
    print('\nSVM Trained Classifier Accuracy for One VS One: ', model_acc)
    print('\nConfusion Matrix for One VS One: \n',conf_mat)
    return(predicted_labels)
```

Question 2.

DAGSVM (One vs One):

In DAGSVM with one vs one strategy is almost same as the one vs one task with one difference that we will take the 1st and last class in the beginning e.g (0,9)

Now, if the predicted values comes out in favor of 0 in that case we will remove the class '9' from the class set for this data point as this data point can never be classified same this class.

```
def dag_one_vs_one(data, svm):

    X_test = data.get('test_samples')
    y_test = data.get('test_samples_labels')
    y_test = y_test.ravel()

    predicted_labels = []*len(y_test)

    for i in range(len(X_test)):
        nums = np.arange(0,10)
        while len(nums) > 1:
            one = nums[0]
            last = nums[len(nums) - 1]
            tt = svm[one][last].predict([X_test[i]],1)

            if tt > 0:
                nums = nums[0:len(nums)-1]
            else:
                nums = nums[1:len(nums)]
        predicted_labels.append(nums[0])

    correct = np.sum(predicted_labels == y_test)
    #print(predicted_labels)
    #print(y_test)
    print(str(correct)+" out of "+str(len(predicted_labels))+ " predictions are correct")
    model_acc = correct/len(predicted_labels)
    conf_mat = confusion_matrix(y_test,predicted_labels)
    print('\nDAG SVM Trained Classifier Accuracy for One VS One: ', model_acc)
    print('\nConfusion Matrix Dag SVM for One VS One: \n',conf_mat)
    return(predicted_labels)
```

Main method:

Please comment and uncomment the kernel_fun while testing the code.

```
if __name__ == "__main__":

    data = loadmat('MNIST_data.mat')
    kernel_fun=polynomial_kernel
    #kernel_fun=polynomial_kernel_dsix
    #kernel_fun=gaussian_kernel

    # get one vs one models
    one_vs_one_models= get_one_vs_one_models(data,kernel_fun)
    # one vs one prediction
    one_vs_one(data, one_vs_one_models)
    # get models for one vs rest
    one_vs_rest_models = get_one_vs_rest_models(data,kernel_fun)
    # predict for one vs rest
    one_vs_rest(data, one_vs_rest_models)

    # dag svm one vs one prediction (uses same modles as one vs one svm)
    dag_one_vs_one(data, one_vs_one_models)
```

Results:

Results for One vs One , One vs Rest and DAGSVM (One vs One) using polynomial_kernel with degree 2.

```
942 out of 1000 predictions are correct

SVM Trained Classifier Accuracy for One VS One:  0.942

Confusion Matrix for One VS One:
 [[ 83   0   1   0   0   1   1   0   0   0]
 [  0 121   0   0   0   0   0   0   1   0]
 [  0   0 107   1   1   0   0   1   2   1]
 [  0   0   1 105   0   6   0   2   1   0]
 [  0   0   0   0 102   0   2   0   0   4]
 [  1   0   0   3   1  84   1   0   2   0]
 [  2   0   0   0   0   1  83   0   1   0]
 [  0   1   2   1   1   0   0  92   1   1]
 [  1   0   2   1   2   0   0   2  77   1]
 [  0   0   0   0   1   0   0   2   1  88]]
937 out of 1000 predictions are correct

SVM Trained Classifier Accuracy for One VS rest:  0.937

Confusion Matrix for One VS rest:
 [[ 82   0   1   1   0   0   1   0   0   1]
 [  0 121   0   0   0   0   0   0   1   0]
 [  0   0 108   0   0   0   0   2   3   0]
 [  0   0   0 104   0   5   2   3   0   1]
 [  0   0   1   0 100   0   1   0   2   4]
 [  1   0   0   2   1  83   1   0   4   0]
 [  2   0   1   0   0   2  82   0   0   0]
 [  1   1   2   2   1   0   0  91   0   1]
 [  1   1   0   3   1   1   0   2  76   1]
 [  0   0   0   0   0   0   0   0   2  90]]
943 out of 1000 predictions are correct

DAG SVM Trained Classifier Accuracy for One VS One:  0.943

Confusion Matrix Dag SVM for One VS One:
 [[ 83   0   1   0   0   1   1   0   0   0]
 [  0 121   0   0   0   0   0   0   1   0]
 [  0   0 105   2   1   1   0   1   2   1]
 [  0   0   0 106   0   6   0   2   1   0]
 [  0   0   0   0 102   0   2   0   0   4]
 [  1   0   0   3   1  85   0   0   2   0]
 [  2   0   0   0   0   1  83   0   1   0]
 [  0   0   0   2   2   0   0  93   1   1]
 [  0   0   1   3   1   0   0   3  77   1]
 [  0   0   0   0   1   0   0   2   1  88]]
```

Results for One vs One , One vs Rest and DAGSVM (One vs One) using polynomial_kernel with degree 6.

```
952 out of 1000 predictions are correct

SVM Trained Classifier Accuracy for One VS One:  0.952

Confusion Matrix for One VS One:
 [[ 84   0   0   0   0   1   1   0   0   0]
 [  0 121   0   0   0   0   0   0   1   0]
 [  0   0 110   0   0   0   0   1   1   1]
 [  0   0   1 107   0   5   0   2   0   0]
 [  0   0   1   0 102   0   1   0   0   4]
 [  1   0   1   2   1  86   0   0   1   0]
 [  2   0   0   0   0   2  82   0   1   0]
 [  0   0   1   1   3   0   0  93   1   0]
 [  1   0   1   2   1   0   0   2  78   1]
 [  0   0   0   0   1   0   0   0   2  89]]
949 out of 1000 predictions are correct

SVM Trained Classifier Accuracy for One VS rest:  0.949

Confusion Matrix for One VS rest:
 [[ 85   0   0   0   0   0   1   0   0   0]
 [  0 121   0   0   0   0   0   0   1   0]
 [  0   0 107   0   0   0   0   3   3   0]
 [  0   0   1 106   0   3   2   1   1   1]
 [  0   0   1   0 102   0   1   0   1   3]
 [  1   0   0   2   1  85   1   0   2   0]
 [  2   0   0   0   0   2  82   0   1   0]
 [  0   1   1   1   2   0   0  94   0   0]
 [  1   0   1   2   1   1   0   1  78   1]
 [  0   0   0   0   1   0   0   0   2  89]]
955 out of 1000 predictions are correct

DAG SVM Trained Classifier Accuracy for One VS One:  0.955

Confusion Matrix Dag SVM for One VS One:
 [[ 84   0   0   0   0   1   1   0   0   0]
 [  0 121   0   0   0   0   0   0   1   0]
 [  0   0 110   0   0   0   0   1   1   1]
 [  0   0   0 108   0   5   0   2   0   0]
 [  0   0   1   0 102   0   1   0   0   4]
 [  1   0   0   2   2  86   0   0   1   0]
 [  2   0   0   0   0   2  82   0   1   0]
 [  0   0   0   1   3   0   0  95   0   0]
 [  1   0   1   2   1   0   0   2  78   1]
 [  0   0   0   0   1   0   0   0   2  89]]
```

Results for One vs One , One vs Rest and DAGSVM (One vs One) using gaussian_kernel with sigma 0.5.

```
950 out of 1000 predictions are correct

SVM Trained Classifier Accuracy for One VS One:  0.95

Confusion Matrix for One VS One:
 [[ 84   0   0   0   0   1   1   0   0   0]
 [  0 121   0   0   0   0   0   0   1   0]
 [  0   0 109   0   0   0   0   1   2   1]
 [  0   0   1 105   0   5   0   3   0   1]
 [  0   0   1   0 103   0   1   0   0   3]
 [  1   0   1   2   1  86   0   0   1   0]
 [  2   0   0   0   0   2  82   0   1   0]
 [  0   0   1   1   3   0   0  93   1   0]
 [  1   0   1   2   1   0   0   2  78   1]
 [  0   0   0   0   1   0   0   0   2  89]]
950 out of 1000 predictions are correct

SVM Trained Classifier Accuracy for One VS rest:  0.95

Confusion Matrix for One VS rest:
 [[ 85   0   0   0   0   0   1   0   0   0]
 [  0 121   0   0   0   0   0   0   1   0]
 [  0   0 107   0   0   0   0   2   4   0]
 [  0   0   0 107   0   3   2   1   1   1]
 [  0   0   1   0 102   0   1   0   1   3]
 [  1   0   0   2   1  85   1   0   2   0]
 [  2   0   0   0   0   2  82   0   1   0]
 [  0   1   1   1   2   0   0  94   0   0]
 [  1   0   1   2   1   1   0   1  78   1]
 [  0   0   0   0   1   0   0   0   2  89]]
953 out of 1000 predictions are correct

DAG SVM Trained Classifier Accuracy for One VS One:  0.953

Confusion Matrix Dag SVM for One VS One:
 [[ 84   0   0   0   0   1   1   0   0   0]
 [  0 121   0   0   0   0   0   0   1   0]
 [  0   0 109   0   0   0   0   1   2   1]
 [  0   0   0 106   0   5   0   3   0   1]
 [  0   0   1   0 103   0   1   0   0   3]
 [  1   0   0   2   2  86   0   0   1   0]
 [  2   0   0   0   0   2  82   0   1   0]
 [  0   0   0   1   3   0   0  95   0   0]
 [  1   0   1   2   1   0   0   2  78   1]
 [  0   0   0   0   1   0   0   0   2  89]]
```

Performance Comparison:-
I have tried using Gaussian function to improve the

	One vs Rest	One vs One	DAGSVM(one vs one)
Polynomial (d = 2)	93.7	94.2	94.3
Polynomial (d = 6)	94.9	95.2	95.5
Gaussian	95	95	95.3

Question 3

I have used the polynomial kernel with degree = 6 with DAGSVM one vs one scheme. The result is better (95.5% accuracy) than the baseline implementation which is 94.3. Whereas, the implementation with Gaussian kernel and sigma = 0.5, I was getting 95.3 accuracy, which is also better than the baseline implementation.

REFERENCES

1. Bishop, Pattern Recognition And Machine Learning - Springer 2006, Chapter 7, Sparse Kernel Machines.
2. https://cvxopt.org/userguide/coneprog.html#quadratic-programming
3. https://courses.csail.mit.edu/6.867/wiki/images/a/a7/Qp-cvxopt.pdf
4. https://courses.media.mit.edu/2006fall/mas622j/Projects/aisen-project/
5. http://www.mit.edu/~9.520/spring09/Classes/multiclass.pdf