91.350.202/580.202 HW #6 Vishal Gadiraju 01672049

Support Vector Machines (SVM):

Support vector machines (SVMs) are a set of supervised learning models used for classification, regression and outliers detection.The goal of an SVM is to find the optimal separating hyperplane which maximizes the margin of the training data.

A Linear SVM finds the optimal hyperplane to separate a dataset containing only 2 dimensions.
The process of learning an SVM (linear classifier with a maximal margin) is the process of solving an optimization problem based on the training set. Thus,we always look for (w, b) which maximizes the margin.
Form of equation defining the decision surface separating the classes is a hyperplane of the form:
$wTx + b = 0$
w is a weight vector
x is input vector and
b is the bias

Quadratic Programming:

Quadratic Programming is an approach used by SVM to solve the optimization problem.
When the objective function is convex (quadratic) and all the constraints are linear, we can solve the problem efficiently using quadratic programming (QP) approach.

IRIS Dataset:

The IRIS dataset contains measurements for 150 iris flowers from three different species.

The three classes in the dataset are:

1. Iris-setosa (n=50)
2. Iris-versicolor (n=50)
3. Iris-virginica (n=50)

And the four features of the Iris dataset are:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm

Source Code: The code was implemented in Python and provided below

```
from sklearn import datasets
import matplotlib.pyplot as plt
from cvxopt import matrix, spmatrix
from cvxopt.solvers import qp
from cvxopt import solvers
import numpy as np

Class SVM():
	def __init__(self, C):
		self.C = C

	def fit(self, data, labels):
		K = data * labels[:, np.newaxis]
		K = np.dot(K, K.T)
		N = len(data)
		solvers.options['show_progress'] = False
		sol  =  qp(matrix(K),  matrix(-np.ones((N,  1))),  matrix(np.vstack((np.eye(N),
-np.eye(N)))),
			matrix(np.vstack((np.ones((N, 1)) * self.C, np.zeros((N, 1))))),
			matrix(labels.reshape(1, N)), matrix(np.zeros(1)))
		print "sol:"
		print sol

		self.alpha = np.array(sol['x']).reshape(N)

		self.support_ = [i for i in xrange(N) if self.alpha[i] > 1e-3]
		self.w = (data * (self.alpha * labels)[:, np.newaxis]).sum(axis=0)
		print "\n weights"
		print self.w
		for i in xrange(N):
			if 0 < self.alpha[i] < self.C:
				self.bias = labels[i] - np.dot(self.w, data[i])
				break
    print "\n bias"
		print self.bias

	def predict(self, data):
		if len(data.shape) <= 1:
			self.predict(data.reshape(1, data.shape[0]))

		return np.sign(np.dot(data, self.w) + self.bias)

	def decision_function(self, data):
		return (np.dot(data, self.w) + self.bias) / np.linalg.norm(self.w)

	def score(self, data, labels):
		pr = self.predict(data)
		correct = 0.
		N = len(data)
		for i in xrange(N):
			correct += 1 if pr[i] * labels[i] > 0 else 0
		return correct / N

def formatIris(dataset):
   X = []
```

```
    y = []
    i = 0
    with open(dataset, 'r') as f:
        for line in f:
            sline = line.replace('\n','')
            t = sline.split(',')
            print t[4]
            if   t[4]=='Iris-versicolor':   val = 1
            elif t[4]=='Iris-virginica':    val =-1
            else: val =0

            print val
           # store sepal and petal lengths, and y value (1, -1)
            X.append([float(t[0])*0.1, float(t[2])*0.1])
            y.append(val)

    return np.array(X), np.array(y)

#load iris dataset
iris = datasets.load_iris()

X = iris.data
y = iris.target
x_data = []
x_data.extend(X[51:61].tolist())
x_data.extend(X[101:111].tolist())

y_temp1 = y[51:61].tolist()
y_temp2 = y[101:111].tolist()
y_data = np.hstack((y_temp1,y_temp2))

X = X.astype(np.double)
y = y.astype(np.double)

x_data = np.array(x_data)
x_data = x_data.astype(np.double)
y_data = y_data.astype(np.double)

clf = SVM(1.0)
clf.fit(x_data,y_data)
```

Output:

```
sol:
{'status': 'optimal', 'dual slack': 9.183863075011808e-11, 'iterations': 7, 'relative gap':
34.50506789848083, 'dual objective': -1.9685298096048445e-09, 'gap': 2.863366831850850 5e-09,
'primal objective': -8.29839500758356e-11, 'primal slack': 6.153958108774931e-12, 's': <40x1
matrix, tc='d'>, 'primal infeasibility': 2.2892940686487102e-11, 'dual infeasibility':
1.0979426536895535e-15, 'y': <1x1 matrix, tc='d'>, 'x': <20x1 matrix, tc='d'>, 'z': <40x1
matrix, tc='d'>}
weights
[ -2.32896974e-10  -8.99622228e-11  -3.25540599e-10  -1.37478513e-10]
```

```
bias
1.00000000345
```

References:

1.http://cs229.stanford.edu/notes/cs229-notes3.pdf

2.Wikipedia- Support Vector Machines

3.http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf