CS 156a Set 8

Cason Shepard

January 18, 2023

problem 1

The SVM problem is a quadratic programming problem. Furthermore, it uses a weight vector w which has $d+1$ weights as we have used in the PLA and Linear Regression.

The answer is [d]

problem 2

When running the algorithm below with C = 0.01 and Q = 2, I got the following E_{in} values:

(a) 0 versus all, $E_{in} = 0.119$

(b) 2 versus all, $E_{in} = 0.100$

(c) 4 versus all, $E_{in} = 0.089$

(d) 6 versus all, $E_{in} = 0.091$

(e) 8 versus all, $E_{in} = 0.074$

The answer is [a]

problem 3

When running the algorithm below with C = 0.01 and Q = 2, I got the following E_{in} values:

(a) 1 versus all, $E_{in} = 0.014$

(b) 3 versus all, $E_{in} = 0.090$

(c) 5 versus all, $E_{in} = 0.076$

(d) 7 versus all, $E_{in} = 0.088$

(e) 9 versus all, $E_{in} = 0.088$

The answer is [a]

problem 4

For 0-versus-all, there were 2279 support vectors. For 1-versus-all, there were 400 support vectors. The difference between these is 1879, which is closest to 1800.

The answer is [c]

problem 5

My algorithm generated the following values. It is apparent that the maximum value for C achieved the lowest E_{in} at 0.003.

C value: 0.001
Support Vectors: 76 E_{in} : 0.004 E_{out} : 0.017

C value: 0.01
Support Vectors: 34 E_{in} : 0.004 E_{out} : 0.019

C value: 0.1
Support Vectors: 24 E_{in} : 0.004 E_{out} : 0.019

C value: 1
Support Vectors: 24E_{in} : 0.003 E_{out} : 0.019

The answer is [d]

problem 6

My algorithm generated the following values when C was 0.001. Notice that when the degree is 5, there are fewer support vectors.

Degree: 2
Support Vectors: 76 E_{in} : 0.004 E_{out} : 0.017

Degree: 5
Support Vectors: 25 E_{in} : 0.004 E_{out} : 0.021

The answer is [b]

problem 7

After running this algorithm 100 times, My algorithm got the following result.

(a) C = 0.0001 was chosen 37 times

(b) C = 0.001 was chosen 53 times

(c) C = 0.01 was chosen 2 times

(d) C = 0.1 was chosen 5 times

(e) C = 1 was chosen 3 times

The answer is [b]

problem 8

I ran for 1000 iterations with C = 0.001, and got an E_{cv} of 0.0048.

The answer is [c]

problem 9

When running my algorithm with RBF kernel, I got the following E_{in} values:

(a) C = 0.01, $E_{in} = 0.0038$

(b) C = 1, $E_{in} = 0.0045$

(c) C = 100, $E_{in} = 0.0032$

(d) C = 10^4, $E_{in} = 0.0026$

(e) C = 10^6, $E_{in} = 0.0006$

The answer is [e]

problem 10

When running my algorithm with RBF kernel, I got the following E_{out} values:

(a) C = 0.01, $E_{out} = 0.0236$

(b) C = 1, $E_{out} = 0.0212$

(c) C = 100, $E_{out} = 0.0189$

(d) C = 10^4, $E_{out} = 0.0236$

(e) C = 10^6, $E_{out} = 0.0236$

The answer is [c]

The following algorithm was used to answer questions 2-4:

```
train_data = []

with open('features.train') as feat:
    for row in feat.readlines():
        row = row [:len(row)-1]
        row = row.split(' ')
        row1 = [x for x in row if x != '']
        row1[0] = float(row1[0])
        row1[1] = float(row1[1])
        row1[2] = float(row1[2])
        train_data.append(row1)

test_data = []

with open('features.test') as feat:
    for row in feat.readlines():
        row = row [:len(row)-1]
        row = row.split(' ')
        row1 = [x for x in row if x != '']
        row1[0] = float(row1[0])
        row1[1] = float(row1[1])
        row1[2] = float(row1[2])
        test_data.append(row1)

def versus_all(n, data_set):
    #splits data and classification info
    data = [[data_set[x][1], data_set[x][2]] for x in range(0, len(data_set))]
    classifier = [1 if x[0] == n else -1 for x in data_set]
    return classifier, data

def count_diff(list_1, list_2):
    count = 0
    for x in range(0, len(list_1)):
        if list_1[x] != list_2[x]:
            count += 1
    return count

from sklearn import svm
for vers in [1, 3, 5, 7, 9]:
    clf = svm.SVC(kernel="poly", C=.01, degree=2, gamma=1)
    true_class, points = versus_all(vers, train_data)
    clf.fit(points, true_class)
    sup_vectors = clf.support_vectors_
    clf_guess = clf.predict(points)
    E_in = count_diff(clf_guess, true_class)/len(true_class)
    print(E_in)
```

The following algorithm was used to answer questions 5-6

```
    # 1 versus 5 Classification
from sklearn import svm

train_1v5 = []
for x in train_data:
    if (x[0] == 1 or x[0] == 5):
        train_1v5.append(x)

test_1v5 = []
class_1v5 = []

for x in test_data:
    if (x[0] == 1 or x[0] == 5):
        test_1v5.append(x[1:])
        if x[0] == 1:
            class_1v5.append(1)
        else:
            class_1v5.append(-1)

for d in [2, 5]:
    clf = svm.SVC(kernel="poly", C=1, degree=d, gamma=1, coef0=1)
    true_class, points = versus_all(1, train_1v5)
    clf.fit(points, true_class)
    sup_vectors = clf.support_vectors_
    E_in = count_diff(clf.predict(points), true_class)/len(true_class)
    E_out = count_diff(clf.predict(test_1v5), class_1v5)/len(class_1v5)
    print("Degree value:" + str(d))
    print("Support Vectors:" + str(len(sup_vectors)))
    print("E_in:"+ str(E_in))
    print("E_out:"+ str(E_out))
```

The following algorithm was used to answer questions 7-8

```
from sklearn import svm
import numpy as np
from random import uniform

train_1v5 = []
for x in train_data:
    if (x[0] == 1 or x[0] == 5):
        train_1v5.append(x)

selection = [0, 0, 0, 0, 0]
avg_E_cv = 0
for i in range(0, 100):
    idx_array = [int(uniform(0, len(train_1v5))) for x in range(0, int(len(train_1v5)/10))]

    true_class, points = versus_all(1, train_1v5)

    fold_train = []
    fold_train_class = []
    fold_validation = []
    fold_validation_class = []

    for idx, point in enumerate(train_1v5):
        if idx in idx_array:
            fold_validation.append(points[idx])
            fold_validation_class.append(true_class[idx])
        else:
            fold_train.append(points[idx])
            fold_train_class.append(true_class[idx])

    cv = []
    c_values =[.0001, .001, .01, .1, 1]
    for c in c_values:
        clf = svm.SVC(kernel="poly", C=c, degree=2, gamma=1, coef0=1)
        clf.fit(fold_train, fold_train_class)
        E_cv = count_diff(clf.predict(fold_validation), fold_validation_class)/len(fold_validation)
        cv.append(E_cv)

    selection[cv.index(min(cv))] += 1

print(selection)
```

The following algorithm as used to answer questions 9-10

```
for c in [0.01, 1, 100, 10000, 1000000]:
    clf = svm.SVC(kernel="rbf", C=c, gamma=1, coef0=1)
    true_class, points = versus_all(1, train_1v5)
    clf.fit(points, true_class)
    E_in = count_diff(clf.predict(points), true_class)/len(true_class)
    E_out = count_diff(clf.predict(test_1v5), class_1v5)/len(class_1v5)
    print("C value:" + str(c))
    print("E_in:"+ str(E_in))
    print("E_out:"+ str(E_out))
```