Eren Barış Bostancı
68770

ENGR 421 – HW6

Data Importing:

I imported the data and divide it to training and test sets.

```
data = np.genfromtxt("hw06_images.csv", delimiter=",")
labels = np.genfromtxt("hw06_labels.csv", dtype=int)

X_train = data[0:1000]
y_train = labels[0:1000]

X_test = data[1000:]
y_test = labels[1000:]

N_train = len(y_train)
D_train = X_train.shape[1]

N_test = len(y_test)
D_test = X_test.shape[1]

C = 10
s = 10
```

```
def gaussian_kernel(X1, X2, s):
    D = dt.cdist(X1, X2)
    K = np.exp(-D ** 2 / (2 * s ** 2))
    return K


K_train = gaussian_kernel(X_train, X_train, s)
```

LAB8 Code Modifications & Confusion Matrixes:

For Lab8 code I put it in a function to call multiple times and this function takes y, C value (for part 7) and s value. Actually, s value is constant for all parts, but I add it to make the function more dynamic. And it returns the alpha and w0 parameters.

```
def lab8(y, C, s):
    yyK = np.matmul(y[:, None], y[None, :]) * K_train

    # set learning parameters
    epsilon = 1e-3

    P = cvx.matrix(yyK)
    q = cvx.matrix(-np.ones((N_train, 1)))
    G = cvx.matrix(np.vstack((-np.eye(N_train), np.eye(N_train))))
    h = cvx.matrix(np.vstack((np.zeros((N_train, 1)), C * np.ones((N_train, 1)))))
    A = cvx.matrix(1.0 * y[None, :])
    b = cvx.matrix(0.0)

    # use cvxopt library to solve QP problems
    result = cvx.solvers.qp(P, q, G, h, A, b)
    alpha = np.reshape(result["x"], N_train)
    alpha[alpha < C * epsilon] = 0
    alpha[alpha > C * (1 - epsilon)] = C

    # find bias parameter
    support_indices, = np.where(alpha != 0)
    active_indices, = np.where(np.logical_and(alpha != 0, alpha < C))
    w0 = np.mean(
        y[active_indices] * (1 - np.matmul(yyK[np.ix_(active_indices, support_indices)], alpha[support_indices])))
    return alpha, w0
```

Since we have 5 classes I create a copy of y_train for 5 times for each class. For each class I set the other class values -1 and set that class's values to 1 for the One-Versus-All approach. For example, for 3rd class in its y array the 1, 2, 4, and 5 values set to -1 and 3 to 1.

```
Y = []
for i in range(5):
    tmp = y_train.copy()
    tmp[tmp != i + 1] = -1
    tmp[tmp == i + 1] = 1
    Y.append(tmp)
```

Then called the lab8 function for all classes to calculate the f_predicteds.

```
for y in Y:
    alpha, w0 = lab8(y, C, s)
    Alphas.append(alpha)
    w0s.append(w0)

    f_predicted = np.matmul(K_train, y[:, None] * alpha[:, None]) + w0
    F.append(f_predicted)
    # calculate confusion matrix
    y_predicted = 2 * (f_predicted > 0.0) - 1
    confusion_matrix = pd.crosstab(np.reshape(y_predicted, N_train), y, rownames=['y_predicted'], colnames=['y_train'])
    print(confusion_matrix)
```

Then combine the results and printed Confusion Matrix for Train data. I calculated the f_predicteds via parameters y, alpha and w0 for each class. Then I take the max values' classes with np.argmax function for each data and used it.

```
preds_train = np.argmax(F, axis=0) + 1
confusion_matrix_train = pd.crosstab(np.reshape(preds_train, N_train), y_train, rownames=['y_predicted'], colnames=['y_train'])
print(confusion_matrix_train)
```

```
Confusion Matrix Train
y_train        1    2    3    4    5
y_predicted
1            207    1    0    9    0
2              2  199    1    1    0
3              0    1  204    6    0
4              0    1    4  185    1
5              0    0    0    0  178
```

The result same with the pdf's.

For test data, again for each data I calculated the f_predicteds for all classes and put them in array (after that I picked the max values' class). For calculating f_predicteds I used the alpha,

w0, and y for each classes own where I obtained while training data part. I didn't recall the Lab8 function again.

```
K_test = gaussian_kernel(X_test, X_train, 10)
F_test = []
f_preds = []

for i in range(len(Y)):
    y = Y[i]
    alpha = Alphas[i]
    w0 = w0s[i]
    f_predicted = np.matmul(K_test, y[:, None] * alpha[:, None]) + w0
    f_preds.append(f_predicted)


preds_test = np.argmax(f_preds, axis=0) + 1
confusion_matrix_test = pd.crosstab(np.reshape(preds_test, N_test), y_test, rownames=['y_predicted'], colnames=['y_test'])
print(confusion_matrix_test)
```

And I found the confusion matrix for test data, and it is same with the pdf's.

```
Confusion Matrix Test
y_test         1    2    3    4    5
y_predicted
1            641   23    3  137    9
2             43  714   27   40    4
3              4   39  666   90   10
4            100   32   69  541   16
5             12    2    6   15  757
```

Part 7 Plotting Accuracy for Different C Values:

I redo the same thing to find the confusion matrixes for different C values for both test and train data and store their matrixes in two different arrays.

```
113  Cs = [0.1, 1, 10, 100, 1000]
114  training_cm = []
115  test_cm = []
116
117
118  for c in Cs:
119      F_train = []
120      Alphas = []
121      w0s = []
122      for y in Y:
123          alpha, w0 = lab8(y, c, s)
124          Alphas.append(alpha)
125          w0s.append(w0)
126
127          f_predicted = np.matmul(K_train, y[:, None] * alpha[:, None]) + w0
128          F_train.append(f_predicted)
129          # calculate confusion matrix
130          y_predicted = 2 * (f_predicted > 0.0) - 1
131          confusion_matrix = pd.crosstab(np.reshape(y_predicted, N_train), y, rownames=['y_predicted'], colnames=['y_train'])
132          print(confusion_matrix)
133
134      preds = np.argmax(F_train, axis=0)
135      preds = preds + 1
136      confusion_matrix = pd.crosstab(np.reshape(preds, N_train), y_train, rownames=['y_predicted'], colnames=['y_train'])
137      print(confusion_matrix)
138      training_cm.append(confusion_matrix)
```

```
    f_test_preds = []

    for i in range(len(Y)):
        y = Y[i]
        alpha = Alphas[i]
        w0 = w0s[i]
        f_predicted = np.matmul(K_test, y[:, None] * alpha[:, None]) + w0
        f_test_preds.append(f_predicted)
    preds_test = np.argmax(f_test_preds, axis=0) + 1
    confusion_matrix = pd.crosstab(np.reshape(preds_test, N_test), y_test, rownames=['y_predicted'], colnames=['y_test'])
    print(confusion_matrix)
    test_cm.append(confusion_matrix)
```

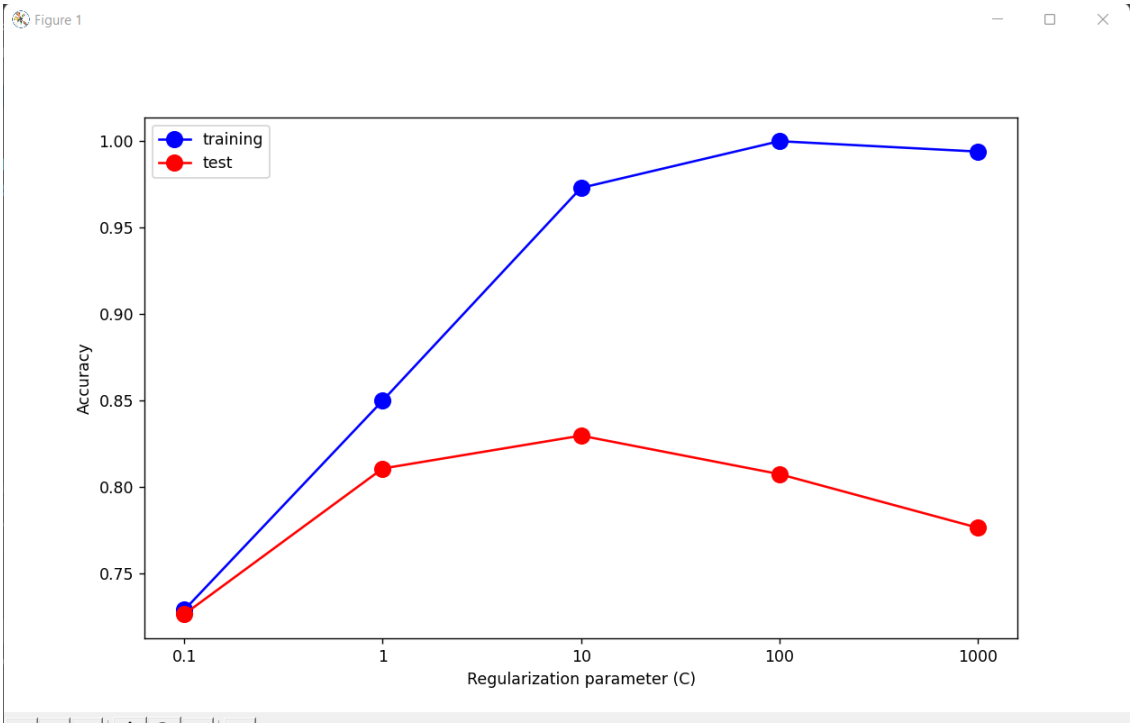
For accuracy calculation, I take the diagonal values' sum and divide it with the total sum of matrix for each of them separately.

```
sum = 0
accur = 0
accuracy_training = []
for cm in training_cm:
    for i in range(5):
        for j in range(5):
            sum = sum + cm.loc[i + 1, j + 1]
            if i == j:
                accur = accur + cm.loc[i + 1, j + 1]
    accuracy_training.append(accur / sum)
    accur = 0
    sum = 0

sum = 0
accur = 0
accuracy_test = []
for cm in test_cm:
    for i in range(5):
        for j in range(5):
            sum = sum + cm.loc[i + 1, j + 1]
            if i == j:
                accur = accur + cm.loc[i + 1, j + 1]
    accuracy_test.append(accur / sum)
    accur = 0
    sum = 0
```

When plotting the plot since in C values interval is not equal but, in the pdf, it was set to equal interval, I convert the C values label to string. The values are exactly same with the pdf's plot.

```
C_values = list(map(str, Cs))
plt.figure(figsize=(10, 6))
plt.plot(C_values, accuracy_training, "bo-", markersize=10, label="training")
plt.plot(C_values, accuracy_test, "ro-", markersize=10, label="test")
plt.xlabel("Regularization parameter (C)")
plt.ylabel("Accuracy")
plt.legend(loc='upper left')
plt.show()
```



At the end, before plotting, since printed confusion matrixes for train and test data at part 5 and part 6 were lost after the cvx.solvers's prints, I reprint the same calculated matrixes. To look at them easily.

```
# Final Prints (I printed matrixes agian since they were lost after cvxopt.solvers' prints)
print("\n")
print("Confusion Matrix Train")
print(confusion_matrix_train)
print("\n")
print("Confusion Matrix Test")
print(confusion_matrix_test)
```