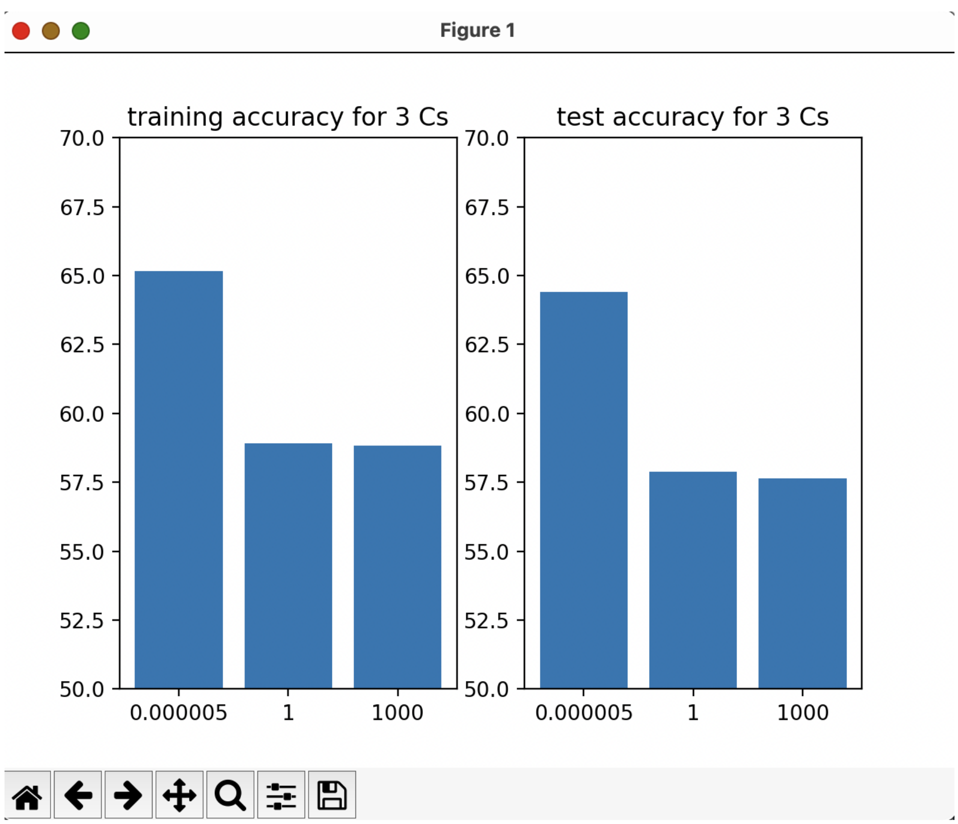
I was initially confused why the SVM dual function K variable went to 2 instead of 785 shown below:

```
def svm_dual(x_batch, y_batch, C):

    n_samples = x_batch.shape[0]
    K = np.zeros((n_samples, n_samples))
    for i in range(n_samples):
        for j in range(n_samples):
            K[i,j] = np.dot(x_batch[i,:2], x_batch[j,:2])
            #K[i,j] = np.dot(x_batch[i,:785], x_batch[j,:785])

    P = cvxopt.matrix(np.outer(y_batch,y_batch) * K)
```

My main function can be seen below. No matter what I tried I could not get my accuracy up to 80 % as indicated in the instructions. I believe my logic and conceptual understanding of support vector machines is there, but I'm not sure my implementation was sound. I also found it very annoying to debug given that the data set was so large it would take me several minutes for the program to finish and I could verify my results. It seemed that for lower values of C, there were less support vectors. And for higher values of C, the program used all 4750 support vectors. This was confusing to me and made me think my program was wrong. (I still think it wrong.). What is also worrying me is that, in your most recent email, you it seems that as we increase C, our accuracy increases which is not what my results indicate.



```python
################################################################################
#                        Main Function
################################################################################
# In the computer assignment, you should complete the main functino by yourself
# In the main function, you need to finish three tasks
# (1) use the one-vs-all method to train SVM models to get the corresponding weights.
# (2) use the one-vs-all method to test the learned SVM modesl to check the learned models.
# (3) obtain the training accuracy and testing accuracy during the above two tasks.
# Note:
# (0) you should install the cvxopt package into your environment, the command is 'conda install -c conda-forge cvxopt'
# (1) you can reuse the code in the svm_example.py file to solve Wolfe Dual problem for learning weights.
# (2) MNIST dataset is non-sperable case, you can refer 3-2 slides Page 7-8 to complete the classifcation.
# (3) you can refer 3-1 slides Page 5-8 to design classifcation accuracy function.
# (4) The training accuracy should be around 80% and the testing accuarcy should be around 60%.
# (5) you can resue the code from Model 2 to solve the one-vs-all problem.
# (6) you should use the pre-defined training dataset and testing dataset to train and to test SVM models



################################################################################
# Training SVM models to get the weigts


def train_one_vs_all_svm(train_data_x, train_data_y, C):
    svm_weights = {}
    for i in range(10):
        print('Training SVM model for digit %d...' % i)
        y = train_data_y[:,i]#.reshape(-1,1)
        x = train_data_x

        alpha = svm_dual(x,y,C)
        # Guarantee the value of alpha being greater than or equal to zero based on the KKT conditons (3.2 Slides, Page 4)
        sv = alpha > 1e-5
        alpha_nz = alpha[sv]
        data_nz = x[sv]
        label_nz = y[sv]

        # Derive the weights of the SVM classifer based on SVM Wolfe dual (3.2 Slides, Page 4)
        n_features = data_nz.shape[1]
        n_samples = data_nz.shape[0]
        beta = np.zeros(n_features)

        print("%d support vectors out of %d points" % (len(alpha_nz), n_samples))

        for j in range(n_samples):
            beta += alpha_nz[j] * label_nz[j] * data_nz[j]

        acc,_ = classifcation_ratio(beta, x, y)
        print('the svm classifcation accuracy is {}'.format(acc*100))
        print(classification_accuracy(beta, x, y))

        svm_weights[str(i)] = beta

    return svm_weights
################################################################################
# Testing SVM models

def test_one_vs_all_svm(test_data_x, test_data_y, svm_weights):
    #take average
    train_accuracy = 0
    test_accuracy = 0
```

```
    for i in range(10):
        train_acc, _ = classifcation_ratio(svm_weights[str(i)], train_data_x, train_data_y[:, i])
        train_accuracy += train_acc
        test_acc, _ = classifcation_ratio(svm_weights[str(i)], test_data_x, test_data_y[:, i])
        test_accuracy += test_acc

    train_accuracy /= 10
    test_accuracy /= 10

    return train_accuracy, test_accuracy

###############################################################################
# Main loop throught different C
train_acc_C = []
test_acc_C = []
C_list = [0.00005, 1, 1000]
for c in C_list:
    print("#################  FOR C = {} ####################### ".format(c))
    svm_weights = train_one_vs_all_svm(train_data_x, train_data_y, c)
    train_accuracy, test_accuracy = test_one_vs_all_svm(test_data_x, test_data_y, svm_weights)
    train_accuracy *=100
    test_accuracy *= 100
    print('Training accuracy: %.2f%%' % (train_accuracy))
    print('Testing accuracy: %.2f%%' % (test_accuracy))
    train_acc_C.append(train_accuracy)
    test_acc_C.append(test_accuracy)

#plot
fig, ax = plt.subplots(1,2)
C = ["0.000005", "1", "1000"]
ax[0].bar(C, train_acc_C)
ax[1].bar(C, test_acc_C)
ax[0].set_title("training accuracy for 3 Cs")
ax[1].set_title("test accuracy for 3 Cs")

ax[0].set_ylim([50,70])
ax[1].set_ylim([50,70])



###############################################################################
# Visualizing five images of the MNIST dataset
fig, _axs = plt.subplots(nrows=1, ncols=5)
axs = _axs.flatten()

for i in range(5):
    axs[i].grid(False)
    axs[i].set_xticks([])
    axs[i].set_yticks([])
    image = data_x[i*10,:784]
    image = np.reshape(image,(28,28))
    aa = axs[i].imshow(image,cmap=plt.get_cmap("gray"))

fig.tight_layout()
plt.show()

"""
C = 0.00005:
Training accuracy: 65.15%
Testing accuracy: 64.40%
```