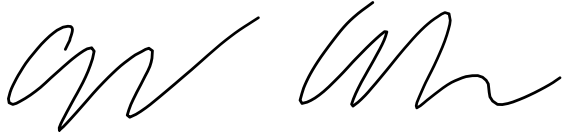


People who helped me on the homework: Madeline (never got last name), attended office hours and had some help from various students.

Honor Code:

"I certify that all solutions are entirely in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted."

Signature:

A handwritten signature in black ink, consisting of two distinct, stylized cursive marks.

Question 2:

Run out of  
time for question  
2 :)

$$2) a) \max_{\lambda_i \geq 0} \min_{\omega, \alpha} \|\omega\|^2 - \sum_{i=1}^n \lambda_i (\gamma_i (X_i \cdot \omega + \alpha) - 1)$$

$$\max_{\lambda_i \geq 0} \sum_{i=1}^n \lambda_i - \sum_{i=1}^n \lambda_i (\gamma_i (X_i \cdot \omega + \alpha) - 1)$$

$$\max_{\lambda_i \geq 0} \sum_{i=1}^n \lambda_i - \sum_{i=1}^n \lambda_i \gamma_i X_i \cdot \omega + \alpha - 1$$

$$\max_{\lambda_i \geq 0} \sum_{i=1}^n \lambda_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \lambda_i \gamma_i X_i \lambda_j \gamma_j X_j$$

$$b) f(x) = \begin{cases} +1 & \text{if } \omega \cdot x + \alpha \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

$$f(x) = \begin{cases} +1 & \text{if } \sum_{i=1}^n \lambda_i \gamma_i X_i \cdot x + \alpha \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

← weights of each  $\lambda, \gamma, x$

$$f(x) = \begin{cases} +1 & \text{if } \alpha + \frac{1}{2} \sum_{i=1}^n \lambda_i \gamma_i X_i \cdot x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

c) Looking at the condition, for all  $i > 0$ , the condition goes to 0, therefore for points corresponding to  $\lambda_i^* > 0$  the other variables  $x_i, X_i, \alpha, \omega$  must result in 0.

d) Support vectors are the only training points needed because they add meaningful information to the training set, while other training sets may not.

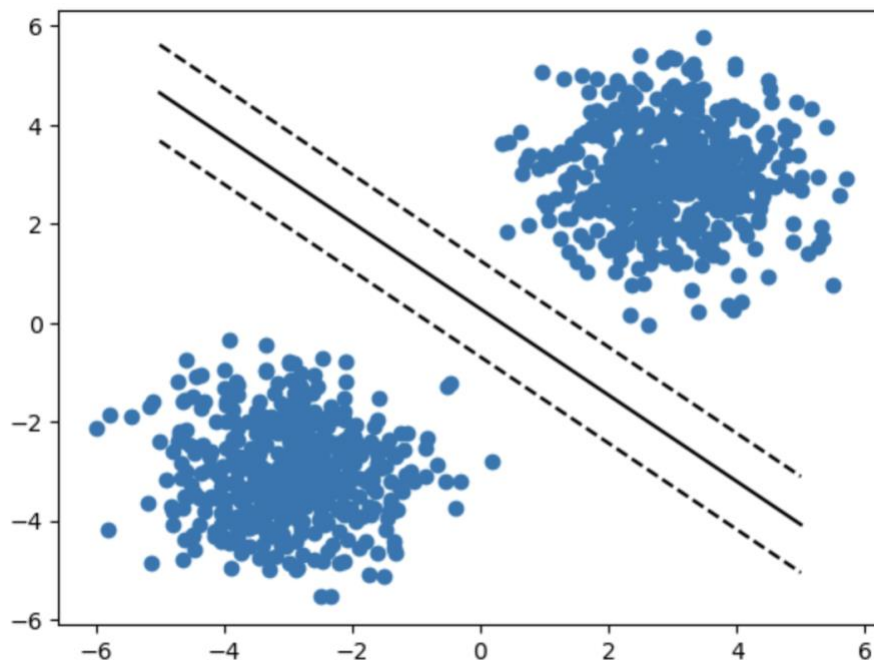
e) The support vectors are the points on the graph closest to the Margin

f) Use contradiction: Let's assume there are no support vectors for each class.

New weight vector:  $\omega' = \frac{\omega}{1 + \epsilon/2}$  with bias  $\alpha'$   $\epsilon > 0$

Symmetric argument: For the class when it does fit, a decision boundary has to be present, per the definition of a support vector, it is the closest to the decision boundary, therefore there must be at least one support vector.

Question 2e Graph:



### Question 3:

```
##Question 3##
def datapartitionMnist():
    mnist_data = np.load('data/mnist-data.npz')

    mnist_training_data = mnist_data['training_data']
    mnist_training_labels = mnist_data['training_labels']
    mnist_test_data = mnist_data['test_data']

    mnist_training_data_flat = mnist_training_data.reshape(mnist_training_data.shape[0], -1)

    mnist_data_combined = np.column_stack((mnist_training_data_flat, mnist_training_labels))

    np.random.shuffle(mnist_data_combined)

    mnist_validation_data = mnist_data_combined[:10000, :-1]
    mnist_validation_labels = mnist_data_combined[:10000, -1]
    mnist_training_data = mnist_data_combined[10000:, :-1]
    mnist_training_labels = mnist_data_combined[10000:, -1]

    return mnist_training_data, mnist_training_labels, mnist_validation_data, mnist_validation_labels

def dataPartitionSpam():
    spam_data = np.load('data/spam-data.npz')

    spam_training_data = spam_data['training_data']
    spam_training_labels = spam_data['training_labels']
    spam_test_data = spam_data['test_data']

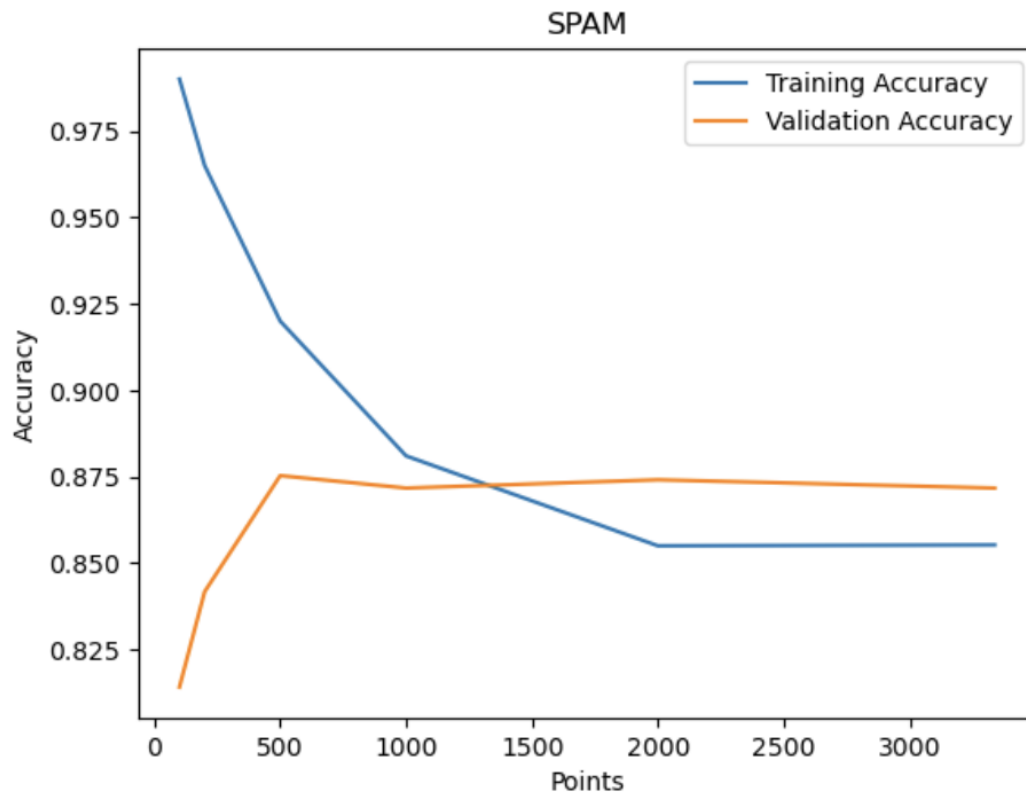
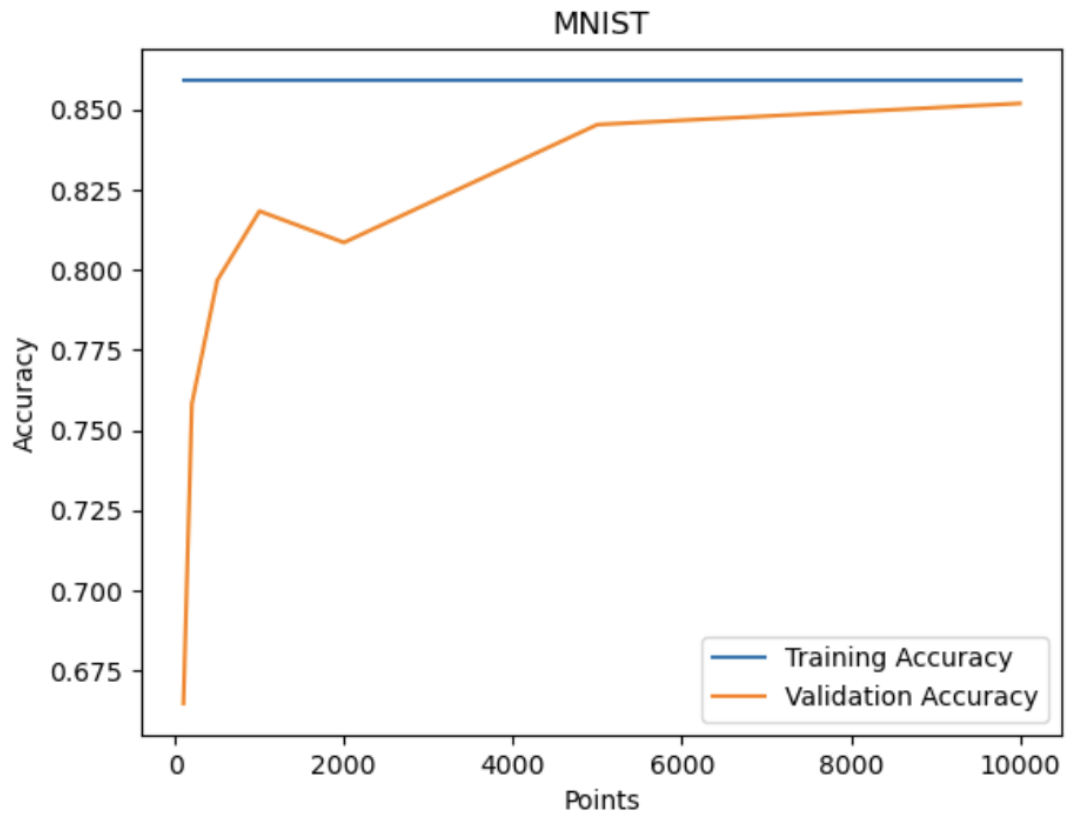
    combined_spam_data = np.column_stack((spam_training_data, spam_training_labels))
    np.random.shuffle(combined_spam_data)
    split_index = int(0.2 * len(combined_spam_data))

    spam_validation_data = combined_spam_data[:split_index, :-1]
    spam_validation_labels = combined_spam_data[:split_index, -1]
    spam_training_data = combined_spam_data[split_index:, :-1]
    spam_training_labels = combined_spam_data[split_index:, -1]

    return spam_training_data, spam_training_labels, spam_validation_data, spam_validation_labels

def accuracy(true_labels, predicted_labels):
    return np.sum((np.array(true_labels) == np.array(predicted_labels)))/len(true_labels)
```

Question 4:



Please reference the code appendix at the bottom of this pdf for the code.

Question 5:

The values I used were: [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]

The accuracy of each value is: [0.8677, 0.8334, 0.8404, 0.8501, 0.8477, 0.8436, 0.8472, 0.8452]

Therefore, the best C value is 0.0001.

Please reference the code appendix at the bottom of this pdf for the code.

Question 6: (Code Appendix)

My C values are: [0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]

The results of those C Values are: [0.7172413793103448, 0.7559220389805097, 0.7991004497751125, 0.8125937031484257, 0.8299850074962519, 0.8410794602698651, 0.8461769115442278, 0.8515742128935532, 0.8233883058470765, 0.7904047976011993, 0.7940029985007495]

Therefore, the best C Value is 10.

Please reference the code appendix at the bottom of this pdf for the code.

Question 7:

Kaggle Score: Mnist: 0.915 Spam: 0.850

What I did to improve my Kaggle Score:

For Mnist, tinkering around with the C values, which I ended up with C=10, I also messed around with other hyperparameters I found in the documentation, but it seems like they only hurt or didn't help the accuracy. The largest bump in accuracy was letting the model train on the full 50,000 points rather than the 10,000 points.

For the Spam dataset, it was stuck for a very long time at around 79%-80% and no hyperparameter was helping, eventually the only thing that brought it up so much was modifying featurize.py, by adding in a lot of words/phrases that I thought were used by spam emails a lot. I ended up looking at a bunch of spam emails.

Please reference the code appendix at the bottom of this pdf for the code.



## Code Appendix:

### Question 4:

```
##Question 4 Mnist##
def train_and_evaluate_svm(training_data, training_labels, validation_data, validation_labels, n):
    training_data_subset = training_data[:n]
    training_labels_subset = training_labels[:n]

    training_data_flat = training_data_subset.reshape(n, -1)
    validation_data_flat = validation_data.reshape(validation_data.shape[0], -1)

    svm = LinearSVC(dual=True)
    svm.fit(training_data_flat, training_labels_subset)

    training_guess = svm.predict(training_data_flat)
    validation_guess = svm.predict(validation_data_flat)

    training_accuracy = accuracy(training_labels_subset, training_guess)
    validation_accuracy = accuracy(validation_labels, validation_guess)

    return training_accuracy, validation_accuracy

a = [100, 200, 500, 1000, 2000, 5000, 10000]

training_data, training_labels, validation_data, validation_labels = datapartitionMnist()

training_accuracies = []
validation_accuracies = []

for i in a:
    training_acc, validation_acc = train_and_evaluate_svm(training_data, training_labels, validation_data, validation_labels, i)
    training_accuracies.append(training_acc)
    validation_accuracies.append(validation_acc)

plt.title('MNIST')
plt.plot(a, training_accuracies, label='Training Accuracy')
plt.plot(a, validation_accuracies, label='Validation Accuracy')
plt.xlabel('Points')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
##Question 4 Spam##
def train_and_evaluate_svm(training_data, training_labels, validation_data, validation_labels, n):
    training_data_subset = training_data[:n]
    training_labels_subset = training_labels[:n]

    training_data_flat = training_data_subset.reshape(n, -1)
    validation_data_flat = validation_data.reshape(validation_data.shape[0], -1)

    svm = LinearSVC(dual=True)
    svm.fit(training_data_flat, training_labels_subset)

    training_predictions = svm.predict(training_data_flat)
    validation_predictions = svm.predict(validation_data_flat)

    training_accuracy = accuracy(training_labels_subset, training_predictions)
    validation_accuracy = accuracy(validation_labels, validation_predictions)

    return training_accuracy, validation_accuracy

a = [100, 200, 500, 1000, 2000, 3337]

train_data, train_labels, validation_data, validation_labels = dataPartitionSpam()

train_accuracies = []
validation_accuracies = []

for i in a:
    train_acc, validation_acc = train_and_evaluate_svm(train_data, train_labels, validation_data, validation_labels, i)
    train_accuracies.append(train_acc)
    validation_accuracies.append(validation_acc)

plt.plot(a, train_accuracies, label='Training Accuracy')
plt.plot(a, validation_accuracies, label='Validation Accuracy')
plt.title('SPAM')
plt.xlabel('Points')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

### Question 5:

```
##Problem 5##
def train_and_evaluate_svm_hyper(training_data, training_labels, validation_data, validation_labels, n):
    flat_training_data = training_data.reshape(training_data.shape[0], -1)
    flat_validation_data = validation_data.reshape(validation_data.shape[0], -1)

    svm = LinearSVC(C=n)
    svm.fit(flat_training_data, training_labels)

    validation_predictions = svm.predict(flat_validation_data)

    validation_accuracy = accuracy(validation_labels, validation_predictions)

    return validation_accuracy

a = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000] # 0.000001
training_data, training_labels, validation_data, validation_labels = datapartitionMnist()
validation_results = []

for i in a:
    validation_acc = train_and_evaluate_svm_hyper(training_data[:10000], training_labels[:10000],
        validation_data, validation_labels, i)

    validation_results.append(validation_acc)

print(validation_results)
```

## Question 6:

```
##Problem 6##
def load_spam_data():
    spam_training_data, spam_training_labels, spam_validation_data, spam_validation_labels = dataPartitionSpam()
    return spam_training_data, spam_training_labels

def k_fold_cross_validation(training_data, training_labels, n): #No need for k
    combined_data = np.column_stack((training_data, training_labels))
    np.random.shuffle(combined_data)

    shuffled_training_data = combined_data[:, :-1]
    shuffled_training_labels = combined_data[:, -1]

    fold_size = len(shuffled_training_data) // 5

    results = []

    for j in n:
        accuracies = []

        for i in range(5):
            validation_data = shuffled_training_data[i * fold_size : (i + 1) * fold_size]
            validation_labels = shuffled_training_labels[i * fold_size : (i + 1) * fold_size]

            training_data = np.concatenate([shuffled_training_data[:i * fold_size], shuffled_training_data[(i + 1) * fold_size :])
            training_labels = np.concatenate([shuffled_training_labels[:i * fold_size], shuffled_training_labels[(i + 1) * fold_size :])

            svm = LinearSVC(C=j, dual=True)
            svm.fit(training_data, training_labels)

            validation_guess = svm.predict(validation_data)

            fold_acc = accuracy(validation_labels, validation_guess)
            accuracies.append(fold_acc)

        avg_accuracy = np.mean(accuracies)

        results.append(avg_accuracy)

    return results

l = [0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
spam_training_data, spam_training_labels = load_spam_data()
cross_validation_results = k_fold_cross_validation(spam_training_data, spam_training_labels, l)
print(cross_validation_results)
```

## Question 7:

```
##Question 7 Mnist DataSet##
def datapartitionMnistKaggle():
    d = 10000

    mnist_data = np.load('data/mnist-data.npz')

    mnist_training_data = mnist_data['training_data']
    mnist_training_labels = mnist_data['training_labels']
    mnist_test_data = mnist_data['test_data']

    mnist_training_data_flat = mnist_training_data.reshape(mnist_training_data.shape[0], -1)

    mnist_data_combined = np.column_stack((mnist_training_data_flat, mnist_training_labels))

    #np.random.shuffle(mnist_data_combined)

    mnist_validation_data = mnist_data_combined[:d, :-1]
    mnist_validation_labels = mnist_data_combined[:d, -1]
    mnist_training_data = mnist_data_combined[d:, :-1]
    mnist_training_labels = mnist_data_combined[d:, -1]

    return mnist_training_data, mnist_training_labels, mnist_validation_data, mnist_validation_labels, mnist_test_data

def train_and_evaluate_svm(training_data, training_labels, validation_data, validation_labels, n, test_data):
    training_data_subset = training_data[:n]
    training_labels_subset = training_labels[:n]

    flat_training_data = training_data_subset.reshape(n, -1)
    flat_validation_data = validation_data.reshape(validation_data.shape[0], -1)

    svm = LinearSVC(C=0.000001, dual=False)
    svm.fit(flat_training_data, training_labels_subset)

    validation_guess = svm.predict(flat_validation_data)

    validation_acc = accuracy_score(validation_labels, validation_guess)
    print("Validation Accuracy:", validation_acc)

    flat_test_data = test_data.reshape(test_data.shape[0], -1)
    results_to_csv(svm.predict(flat_test_data))

training_data, training_labels, validation_data, validation_labels, test_data = datapartitionMnistKaggle()
train_and_evaluate_svm(training_data, training_labels, validation_data, validation_labels, 50000, test_data)

print("Done")
```

##Question 7 Spam##

```
def dataPartitionSpamKaggle():
    spam_data = np.load('data/spam-data.npz')

    spam_training_data = spam_data['training_data']
    spam_training_labels = spam_data['training_labels']
    spam_test_data = spam_data['test_data']

    combined_spam_data = np.column_stack((spam_training_data, spam_training_labels))
    np.random.shuffle(combined_spam_data)
    split = int(0.2 * len(combined_spam_data))

    spam_validation_data = combined_spam_data[:split, :-1]
    spam_validation_labels = combined_spam_data[:split, -1]
    spam_training_data = combined_spam_data[split:, :-1]
    spam_training_labels = combined_spam_data[split:, -1]

    return spam_training_data, spam_training_labels, spam_validation_data, spam_validation_labels, spam_test_data

def train_and_evaluate_svm_Spam(training_data, training_labels, validation_data, validation_labels, test_data):
    flat_training_data = training_data.reshape(training_data.shape[0], -1)
    flat_validation_data = validation_data.reshape(validation_data.shape[0], -1)
    flat_test_data = test_data.reshape(test_data.shape[0], -1)

    svm = LinearSVC(C=10, dual=False)

    svm.fit(flat_training_data, training_labels)

    validation_guess = svm.predict(flat_validation_data)

    validation_acc = accuracy_score(validation_labels, validation_guess)
    print("Validation Accuracy:", validation_acc)

    results_to_csv(svm.predict(flat_test_data))

training_data, training_labels, validation_data, validation_labels, test_data = dataPartitionSpamKaggle()
train_and_evaluate_svm_Spam(training_data, training_labels, validation_data, validation_labels, test_data)
```

Some websites I went to for help:

- <https://numpy.org/doc/stable/reference/index.html>
  - Went to this website a lot to look up numpy functions.
- <https://www.activecampaign.com/blog/spam-words>
  - This was looking for finding out notable key words to filter out of the spam email points.