CS 189/289A Introduction to Machine Learning
Spring 2025 Jonathan Shewchuk HW1

Due: Wednesday, January 29 at 11:59 pm

This homework comprises a set of coding exercises and a few math problems. While we have you train models across three datasets, the code for this entire assignment can be written in under 250 lines. Start this homework early! You can submit to Kaggle only twice a day.

Deliverables:

1. Submit your predictions for the test sets to Kaggle as early as possible. Include your Kaggle scores in your write-up (see below).

2. Submit a **PDF** of your homework, **with an appendix listing all your code**, to the Gradescope assignment entitled "HW1 Write-Up". You may typeset your homework in LaTeX or Word or submit neatly handwritten and scanned solutions. **Please start each question on a new page.** If there are graphs, include those graphs in the correct sections. **Do not** put them in an appendix. We need each solution to be self-contained on pages of its own.

   - On the first page of your write-up, please list students who helped you or whom you helped on the homework. (Note that sending each other code is not allowed.)
   - On the first page of your write-up, please copy the following statement and sign your signature next to it. (Mac Preview, PDF Expert, and FoxIt PDF Reader, among others, have tools to let you sign a PDF file.) We want to make *extra* clear the consequences of cheating.

     "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."

3. Submit all the code needed to reproduce your results to the Gradescope assignment entitled "HW1 Code". You must submit your code twice: once in your PDF write-up (above) so the readers can easily read it, and again in compilable/interpretable form so the readers can easily run it. **Do NOT include any data files we provided.** Please include a short file named README listing your name, student ID, and instructions on how to reproduce your results. Please take care that your code doesn't take up inordinate amounts of time or memory.

   The Kaggle score will not be accepted if the code provided a) does not compile or b) compiles but does not produce the file submitted to Kaggle.

Python Configuration and Data Loading

This section is only setup and requires no submitted solution. Please follow the instructions below to ensure that your Python environment is configured properly, and you are able to successfully load the data provided with this homework. For all coding questions, we recommend using Anaconda for Python 3.

(a) Either install Anaconda for Python 3, or ensure you're using Python 3. To ensure you're running Python 3, open a terminal in your operating system and execute the following command:

```
python --version
```

Do not proceed until you're running Python 3.

(b) Install the following dependencies required for this homework by executing the following command in your operating system's terminal:

```
pip install scikit-learn scipy numpy matplotlib
```

Please use Python 3 with the modules specified above to complete this homework.

(c) You will be running out-of-the-box implementations of Support Vector Machines to classify three datasets. You will find a set of `.npz` files in the `data` folder for this homework. Each `.npz` file will load as a Python dictionary. Each dictionary contains three fields:

- **training_data**, the training set features. Rows are sample points and columns are features.
- **training_labels**, the training set labels. Rows are sample points. There is one column: the labels corresponding to rows of training_data above.
- **test_data**, the test set features. Rows are sample points and columns are features. You will fit a model to predict the labels for this test set, and submit those predictions to Kaggle.

The three datasets for the coding portion of this assignment are described below.

- **toy-data.npz** is a synthetic dataset with two features (2-dimensional) and two classes. The training set has 1,000 examples, and no test set is provided. This dataset is only used in Section 2 of this homework.
- **mnist-data.npz** contains data from the MNIST dataset. There are 60,000 labeled images of handwritten digits for training, and 10,000 for testing. The images are grayscale and provided by 28×28 pixels. There are 10 possible labels for each image: the digits 0–9. We will use the simplest of features for classification: raw pixel brightness values. In other words, our feature vector for an image will be a row vector with all the pixel values concatenated in a row major (or column major) order.

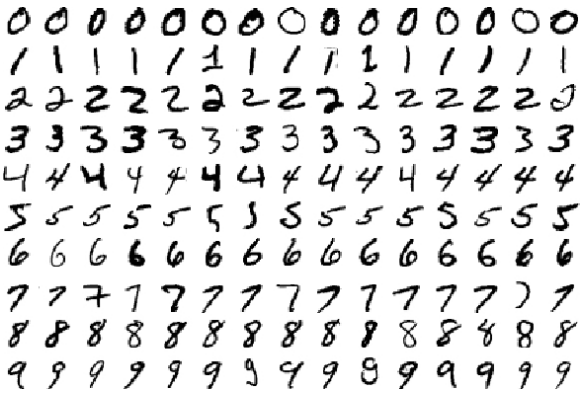


Figure 1: Examples from the MNIST dataset.

- **spam-data.npz** contains email data. The labels are 1 for spam and 0 for ham (not spam). We provide the raw email data in the subfolders **spam**, **ham**, and **test** (unlabeled test data) in the `data` folder. We also provide the script **featurize.py** to compute the frequency of certain words within the email text and generate a corresponding feature vector. You may modify **featurize.py** to generate new features for the email data.

To check whether your Python environment is configured properly for this homework, run the `load` script (copied below) from within the scripts folder. This script should load each dataset and print the shapes of the training data, training labels, and test data. Confirm that the shapes align with your expectations based on the descriptions above.

Pay attention to errors raised when attempting to import any dependencies. Resolve such errors by manually installing the required dependency (e.g. execute `pip install numpy` for import errors relating to the numpy package). You must have your environment configured properly before moving on.

```
import sys
if sys.version_info[0] < 3:
  raise Exception("Python 3 not detected.")
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from scipy import io

if __name__ == "__main__":
    for data_name in ["toy", "mnist", "spam"]:
        data = np.load(f"../data/{data_name}-data.npz")
        print("\nloaded %s data!" % data_name)
        fields = "training_data", "training_labels", "test_data"
        for field in fields:
            print(field, data[field].shape)
```

1 Honor Code

Declare and sign the following statement:

"I certify that all solutions in this document are entirely my own and that I have not looked at anyone else's solution. I have given credit to all external sources I consulted."

Signature : 

While discussions are encouraged, *everything* in your solution must be your (and only your) creation. Furthermore, all external material (i.e., *anything* outside lectures and assigned readings, including figures and pictures) should be cited properly. We wish to remind you that consequences of academic misconduct are *particularly severe*!

This section provides background information on Support Vector Machines (SVMs) used in this homework. You can choose to focus on the coding sections first and revisit this section later, but make sure that this section precedes the coding questions in your write-up.

2 Theory of Hard-Margin Support Vector Machines

A *decision rule* (or *classifier*) is a function $r : \mathbb{R}^d \to \pm 1$ that maps a feature vector (test point) to $+1$ ("in class") or -1 ("not in class"). The decision rule for linear SVMs is of the form

$$r(x) = \begin{cases} +1 & \text{if } w \cdot x + \alpha \geq 0, \\ -1 & \text{otherwise,} \end{cases} \tag{1}$$

where $w \in \mathbb{R}^d$ and $\alpha \in \mathbb{R}$ are the parameters of the SVM. To select the best parameters for this decision rule, suppose we have a set of n training points and correponding labels. Let $X_i \in \mathbb{R}^d$ be the ith training point with corresponding label $y_i \in \{+1, -1\}$. The primal hard-margin SVM optimization problem (used to find the optimal decision rule parameters) is

$$\min_{w,\alpha} \ \|w\|^2 \ \text{ subject to } \ y_i(X_i \cdot w + \alpha) \geq 1, \ \ \forall i \in \{1, \ldots, n\}, \tag{2}$$

where $\|\cdot\|$ denotes the ℓ_2 norm (i.e., $\|w\| = \sqrt{w \cdot w}$).

We can rewrite this optimization problem by using Lagrange multipliers to eliminate the constraints. (If you're curious to know what Lagrange multipliers are, we recommended taking a look at this Wikipedia page, but you don't need to understand them to do this problem.) We thereby obtain the equivalent optimization problem

$$\max_{\lambda_i \geq 0} \min_{w,\alpha} \ \|w\|^2 - \sum_{i=1}^{n} \lambda_i (y_i(X_i \cdot w + \alpha) - 1). \tag{3}$$

Note: λ_i must be greater than or equal to 0.

(a) Show that equation (3) can be rewritten as the *dual optimization problem*

$$\max_{\lambda_i \geq 0} \ \sum_{i=1}^{n} \lambda_i - \frac{1}{4} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j X_i \cdot X_j \ \text{ subject to } \ \sum_{i=1}^{n} \lambda_i y_i = 0. \tag{4}$$

Hint: Use calculus to determine and prove what values of w and α optimize equation (3). Explain where the new constraint comes from.

(b) Suppose we know the values λ_i^* and α^* that optimize equation (3). Show that the decision rule specified by equation (1) can be written

$$r(x) = \begin{cases} +1 & \text{if } \alpha^* + \frac{1}{2} \sum_{i=1}^{n} \lambda_i^* y_i X_i \cdot x \geq 0, \\ -1 & \text{otherwise.} \end{cases} \tag{5}$$

(c) Applying Karush–Kuhn–Tucker (KKT) conditions (see this Wikipedia page for more information), any pair of optimal primal and dual solutions w^*, α^*, λ^* for a linear, hard-margin SVM must satisfy the following condition:

$$\lambda_i^*(y_i(X_i \cdot w^* + \alpha^*) - 1) = 0 \quad \forall i \in \{1, \ldots, n\}$$

This condition is called *complementary slackness*. Explain what this implies for points corresponding to $\lambda_i^* > 0$. That is, what relationship must exist between the data points, labels, and decision rule parameters? (You can provide a one sentence answer.)

(d) The training points X_i for which $\lambda_i^* > 0$ are called the *support vectors*. In practice, we frequently encounter training data sets for which the support vectors are a small minority of the training points, especially when the number of training points is much larger than the number of features. Explain why the support vectors are the only training points needed to use the decision rule.

(e) The obtained parameters when fitting the linear SVM to the 2D synthetic dataset found in **toy-data.npz** approximately correspond to

$$w = \begin{bmatrix} -0.4528 \\ -0.5190 \end{bmatrix} \quad \text{and} \quad \alpha = 0.1471. \tag{6}$$

Using only matplotlib basic plotting functions, produce a plot of

- the data points,
- the decision boundary,
- the margins, defined as $\{x \in \mathbb{R}^2 : w \cdot x + \alpha = \pm 1\}$.

Clearly indicate the points in your plot that are support vectors.

Hint: You can use the functions in the code snippet below, which plot the data points and decision boundary but not the margins.

```
# Plot the data points
def plot_data_points(data, labels):
    plt.scatter(data[:, 0], data[:, 1], c=labels)

# Plot the decision boundary
def plot_decision_boundary(w, b):
    x = np.linspace(-5, 5, 100)
    y = -(w[0] * x + b) / w[1]
    plt.plot(x, y, 'k')

# Plot the margins
def plot_margins(w, b):
    ## TODO
```

(f) Assume the training points are linearly separable. Using the original SVM formulation in equation 2, prove that there is at least one support vector for each class, $+1$ and -1.

Hint: Use contradiction. Construct a new weight vector $w' = w/(1 + \epsilon/2)$ and corresponding bias α' where $\epsilon > 0$. It is up to you to determine what ϵ should be based on the contradiction. If you provide a symmetric argument, you need only provide a proof for one of the two classes.

2) Theory SVM

a) EQ 3 rewritten dual optimization problem

$$\max_{\lambda_i \geq 0} \min_{w,\alpha} \|w\|^2 - \sum_{i=1}^{n} \lambda_i (y_i (x_i \cdot w + \alpha) - 1)$$

Derivation in respect to w and setting to 0

$$w' = 2w - \sum_{i=1}^{n} \lambda_i y_i x_i$$

$$0 = 2w - \sum_{i=1}^{n} \lambda_i y_i x_i$$

$$w = \frac{1}{2} \sum_{i=1}^{n} \lambda_i y_i x_i$$

Derivation in respect to α and setting to 0

$$\alpha' = -\sum_{i=1}^{n} \lambda_i y_i$$

$$0 = \sum_{i=1}^{n} \lambda_i y_i$$

* I know the discovered critical point of the function is a min examining the convexity of the function. Term $\|w\|^2$ is a quadratic function w/ a positive coefficient, telling me in that our critical point is likely a min. Additionally, the function is linear in α leading to a convex in α, likely relating to a min

Sub backs:

$$\|w\|^2 = \left(\frac{1}{2} \sum_{i=1}^{n} \lambda_i y_i x_i \right)^2 = \frac{1}{4} \sum_{i=1}^{n} \sum_{s=1}^{n} \lambda_i \lambda_s y_i y_s (x_i x_s)$$

$$\max_{\lambda_i \geq 0} \min_{w, \alpha} \|w\|^2 - \sum_{i=1}^{n} \lambda_i (y_i (x_i \cdot w + \alpha) - 1)$$

$$\rightarrow \max_{\lambda_i \geq 0} \min_{w, \alpha} \frac{1}{4} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j (x_i x_j) - \sum_{i=1}^{n} \lambda_i (y_i (x_i \cdot w + \alpha) - 1)$$

$$\sum_{i=1}^{n} \lambda_i y_i x_i w + \lambda_i y_i \alpha - \lambda_i$$

$\lambda_i y_i x_i w = 0$ $\quad$ $\lambda_i y_i \alpha = 0$

due to $\sum_{i=1}^{n} \lambda_i y_i = 0$

$$\rightarrow \max_{\lambda_i \geq 0} \sum_{i=1}^{n} \lambda_i - \frac{1}{4} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j (x_i x_j)$$

b) Eq 1:

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

Sub in known values of w from a)

$$\rightarrow r(x) = \begin{cases} +1 & \text{if } \alpha^* + \frac{1}{2} \sum_{i=1}^{n} \lambda_i y_i x_i \cdot x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

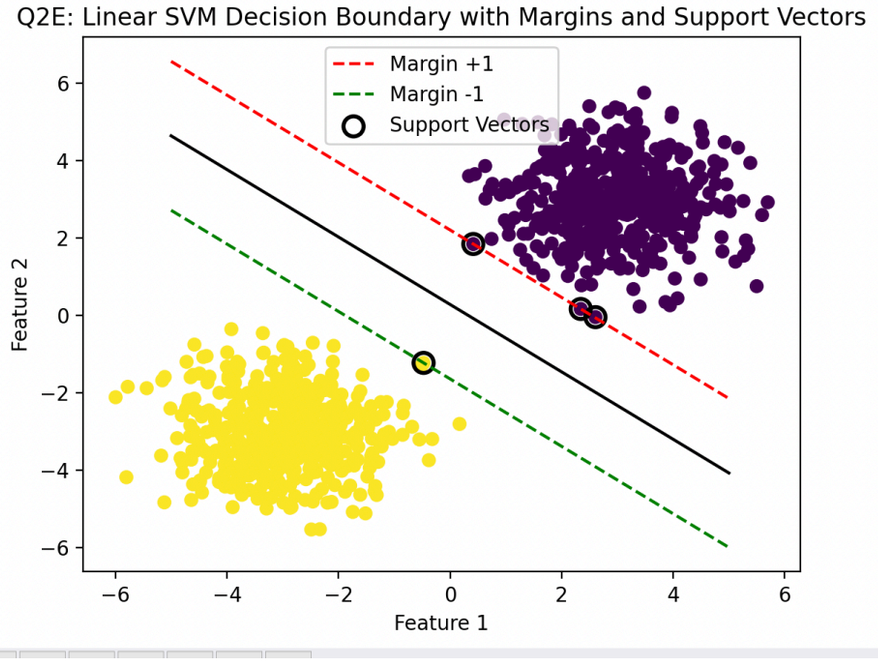
c) The optimal solution needs to fulfill the condition $y_i(x_i \cdot w + \alpha) \geq 1$ based on EQ2.

↳ thus if $\lambda_i^* > 0$, then $y_i(x_i \cdot w^* + \alpha^*) - 1$ must equal 0 for complementary slackness to hold

* $\lambda_i^* > 0$ only for points on the margin

d) Support vectors are the only training points needed because they are the only training points making contribution. Training points not in support vector has $\lambda_i^* = 0$. Hard margin SVM must classify that training point in the support vector correctly

e)



```python
# #Question 2E Plot Code

# Parameters for w & alpha + load-toy data specific to training data
w = np.array([-0.4528, -0.5190])
alpha = 0.1471
data = np.load("/Users/dankim/Downloads/Cal Stuff/Cal Coursework F2020-Sp2025/Cal Berkeley Spring 2025/CS 189/Homework/hw1/data/toy-data.npz")
training_data = data["training_data"]
decision_values = np.dot(training_data, w) + alpha
labels = np.sign(decision_values)


# Support Vectors
pre_support_vector = np.isclose(np.abs(decision_values), 1, atol=0.05)
support_vectors = training_data[pre_support_vector]


# Plot the data points
def plot_data_points(training_data, labels):
    plt.scatter(training_data[:, 0], training_data[:, 1], c=labels)

# Plot the decision boundary
def plot_decision_boundary(w, b):
    x = np.linspace(-5, 5, 100)
    y =-(w[0] * x + b) / w[1]
    plt.plot(x, y, 'k')

#Plot the margins
def plot_margins(w, b):
    x = np.linspace(-5, 5, 100)
    y1 = -(w[0] * x + b + 1) / w[1]
    y2 = -(w[0] * x + b - 1) / w[1]
    plt.plot(x, y1, 'r--', label="Margin +1")
    plt.plot(x, y2, 'g--', label="Margin -1")

# Plot complete Graph
plot_data_points(training_data, labels)
plot_decision_boundary(w, alpha)
plot_margins(w, alpha)

#C Highlight support vectors
plt.scatter(support_vectors[:, 0], support_vectors[:, 1], s=100, facecolors='none', edgecolors='k', linewidths=2, label="Support Vectors")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.title("Q2E: Linear SVM Decision Boundary with Margins and Support Vectors")
plt.show()
```

f) Prove at least 1 support vector for each class ± 1

Considering $+1$ class

No support vector missing even though point is beyond margin for all i w/ $y_i = +1$.

If $z > 0$ we shifted Distance from nearest N points:

$$z = \min x_i \cdot w + \alpha - 1$$

To show this:

Construct a support vector by making the point closest to constraint Boundary x_i one.

Hint: $w' = w/(1 + z/2)$

Find α' w/ conditions a) $x_j \cdot w' + \alpha' = 1$

b) $x_j \cdot w + \alpha = 1 + z$

Let $\alpha' = 1 - x_j \cdot w'$

$$= 1 - \frac{w_j \cdot w}{1 + z/2} = \frac{1 + \frac{z}{2}}{1 + z/2} - \frac{1 + z - \alpha}{1 + z/2}$$

$$= \frac{\alpha - z/2}{1 + z/2}$$

For the entire assignment, you may use sklearn only for the SVM model. Everything else must be done without the use of sklearn.

3 Data Partitioning and Evaluation Metrics

In machine learning, it is typical to rely on a set of held-out data points, referred to as the *validation* set, to evaluate the performance of various models and ultimately select the best performing one, while using the rest of the data, or *training* set, to train the models. In its simplest form, evaluating a trained model requires you to (i) set aside a validation set and (ii) select a reasonable metric to evaluate model performance.

In this question, you will implement these components that will be useful for the rest of the assignment. **Please do not use any sklearn functions in this section**.

(a) **Data partitioning**: Rarely will you receive "training" data and "validation" data; usually you will have to partition available labeled data yourself. In this question, you will *shuffle and partition* each of the datasets in the assignment[1]. Shuffling prior to splitting crucially ensures that all classes are represented in your partitions. For this question, please do not use any functions available in sklearn. For the MNIST dataset, write code that sets aside 10,000 training images as a validation set. For the spam dataset, write code that sets aside 20% of the training data as a validation set.

(b) **Evaluation metric**: There are several ways to evaluate models. We will use *classification accuracy*, or the percent of examples classified correctly, as a measure of the classifier performance. Error rate, or one minus the accuracy, is another common metric. Suppose you have a set of n input observations. Write a function that takes as inputs the set of true labels, $y \in \mathbb{R}^n$, and the set of predicted labels, $\hat{y} \in \mathbb{R}^n$, and computes the classification accuracy score

$$s = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\left[y_i = \hat{y}_i\right]. \tag{7}$$

In the accuracy score, $\mathbb{I}\left[y_i = \hat{y}_i\right]$ is an indicator function defined as

$$\mathbb{I}\left[y_i = \hat{y}_i\right] = \begin{cases} 1 & \text{if } y_i = \hat{y}_i \\ 0 & \text{otherwise} \end{cases}. \tag{8}$$

Here, y_i and $\hat{y}_i$ respectively denote the ground-truth and predicted label for observation i.

Deliverable: Attach a copy of your data partitioning and evaluation metric code to your homework report under question 3.

[1] Make sure that you shuffle the labels with the training images. It's a very common error to mislabel the training images by forgetting to permute the labels with the images!

```python
# Question 3 code: Data Partitioning and Evaluation Metrics

# Question 3A: Data Partitioning for MNIST and spam
def data_partitioning(data, labels, num_vals):
    #Randomness Generator
    num_total = len(data)

    assert num_total == len(labels), "Incorrect Data Size"
    assert num_vals >= 0, "Partion?"

    #For spam data partioning since its a decimal 0.2 instead of a value like 10k
    if num_vals < 1:
        num_vals = int(num_total * num_vals)

    random_index = np.random.permutation(num_total)

    #Training Indeces
    train_indices = random_index[num_vals:]
    val_indices = random_index[:num_vals]

    #Training Data/Labels
    train_data = data[train_indices]
    train_labels = labels[train_indices]

    #Validation Data/Labels
    validation_data = data[val_indices]
    validation_labels = labels[val_indices]

    #Return Training Data/Labels & Validation Data/Labels
    return train_data, train_labels, validation_data, validation_labels


# Question 3B: Evaluation Metric
def evaluation_metric(y_vals, y_predictions):
    return np.mean(y_vals == y_predictions)
```

4 Support Vector Machines: Coding

We will use linear Support Vector Machines (SVMs) to classify our datasets. Train a linear SVM on the MNIST and spam datasets that you divided into training and validation sets in the previous problem. For each dataset, plot the classification accuracy (as defined in the previous problem) on the training and validation sets versus the number of training examples that you used to train your classifier. The number of training examples to use are listed for each dataset in the following parts. To evaluate validation accuracy, you should always use the same validation set obtained in the previous problem. To evaluate training accuracy, you should use the portion of the full training set that is actually used to train the model.

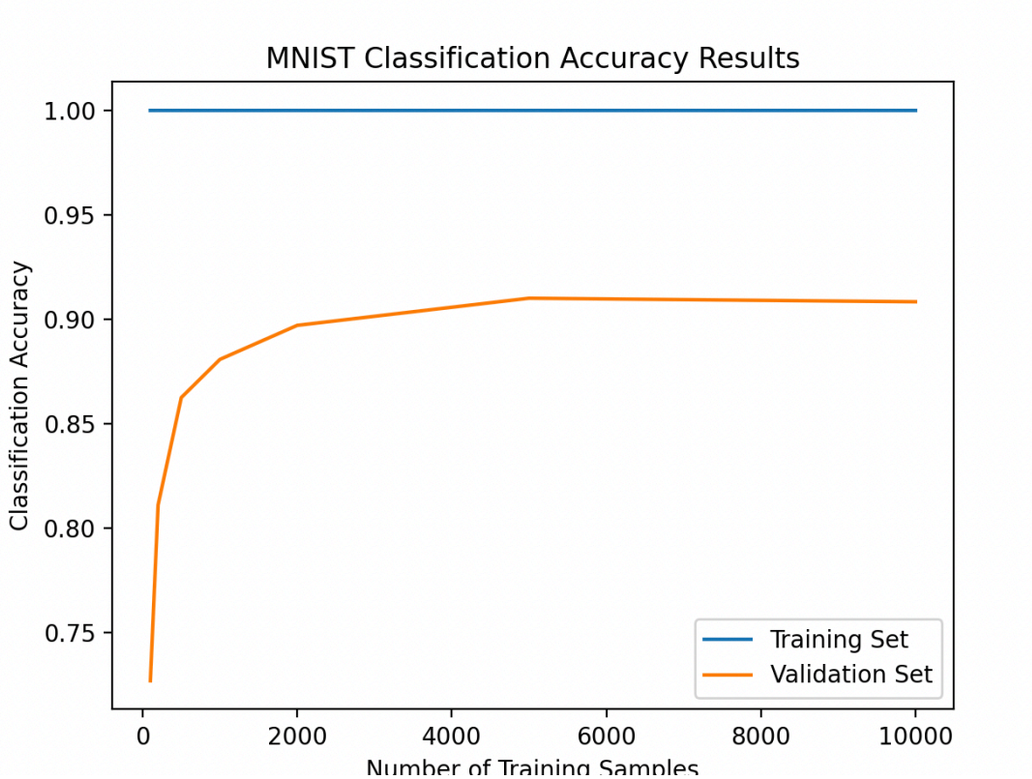
You may use `sklearn` **only** for the SVM model. Everything else must be done without the use of `sklearn`. **Note:** You can use either `SVC(kernel='linear')` or `LinearSVC` as your SVM model, though they each solve slightly different optimization problems using different libraries.

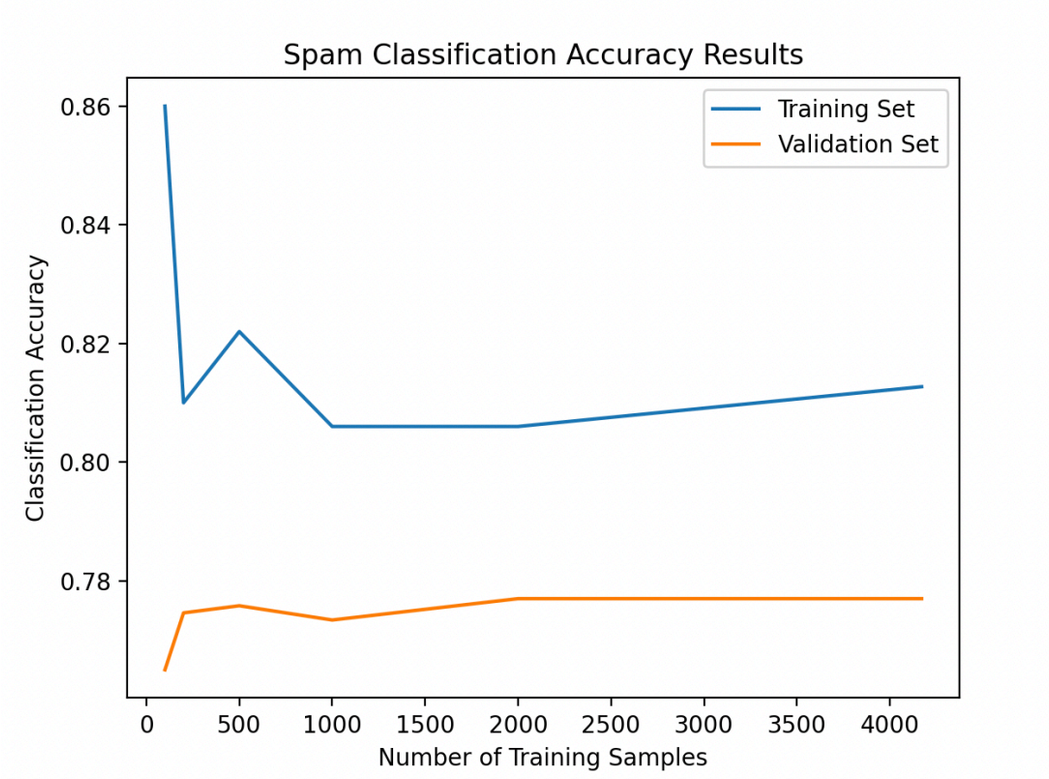
(a) For the **MNIST** dataset, use raw pixels as features. Train your model with the following numbers of training examples: 100, 200, 500, 1,000, 2,000, 5,000, 10,000. You should expect validation accuracies between 70% and 90%. [2]

(b) For the **spam** dataset, use the provided word frequencies as features. Train your model with the following numbers of training examples: 100, 200, 500, 1,000, 2,000, **ALL**. (**ALL** is all of the data we already set aside for training.) Performance may vary some for this dataset, but you should expect validation accuracies between 70% and 90%.

Deliverable: For this question, you should include two plots showing training and validation accuracy versus number of examples for each of the datasets. Additionally, be sure to include your code in the "Code Appendix" portion of your write-up.

[2] Hint: Be consistent with any preprocessing you do. Use either integer values between 0 and 255 or floating-point values between 0 and 1. Training on floats and then testing with integers is bound to cause trouble.


MNIST Classification Accuracy Results
1.00
0.95
0.90
0.85
0.80
0.75
Classification Accuracy
0
2000
4000
6000
8000
10000
Number of Training Samples
Training Set
Validation Set


Spam Classification Accuracy Results
0.86
0.84
0.82
0.80
0.78
Classification Accuracy
0
500
1000
1500
2000
2500
3000
3500
4000
Number of Training Samples
Training Set
Validation Set

```python
# Question 4: Support Vector Machines: Coding
def plot_classification_accuracy(trained_size, train_accuracy, validation_accuracy, data_title):
    plt.title(data_title)
    plt.plot(trained_size, train_accuracy, label="Training Set")
    plt.plot(trained_size, validation_accuracy, label="Validation Set")
    plt.xlabel("Number of Training Samples")
    plt.ylabel("Classification Accuracy")
    plt.legend()
    plt.show()

def SVM_training(train_data, train_labels, validation_data, validation_labels):
    svm_model = SVC(kernel="linear")
    svm_model.fit(train_data, train_labels)

    train_predictions = svm_model.predict(train_data)
    validation_predictions = svm_model.predict(validation_data)

    train_accuracy = evaluation_metric(train_labels, train_predictions)
    validation_accuracy = evaluation_metric(validation_labels, validation_predictions)

    return train_accuracy, validation_accuracy

def individual_SVM_training(total_list, train_data, train_label, validation_data, validation_label):
    list1 = []
    list2 = []
    for size in total_list:
        train_acc, validation_acc = SVM_training(train_data[:size], train_label[:size], validation_data, validation_label)
        list1.append(train_acc)
        list2.append(validation_acc)
    return list1, list2
```

```python
#spam Plot
spam_loaded_data = np.load("/Users/dankim/Downloads/Cal Stuff/Cal Coursework F2020-Sp2025/Cal Berkeley Spring 2025/CS 189/Homework/hw1/data/spam-data.npz")
s_training_data = spam_loaded_data["training_data"]
s_training_labels = spam_loaded_data["training_labels"]
s_num_vals = 0.2

mss_training_size = [100, 200, 500, 1000, 2000, s_training_data.shape[0]]

sM_train_data, sM_train_labels, sM_validation_data, sM_validation_labels = data_partitioning(s_training_data, s_training_labels, s_num_vals)

#Train SVC Model for SVM
s_train_plt_data, s_validation_plt_data = individual_SVM_training(mss_training_size, sM_train_data, sM_train_labels, sM_validation_data, sM_validation_labels)

#Plot Spam
spam_plot = plot_classification_accuracy(mss_training_size, s_train_plt_data, s_validation_plt_data, "Spam Classification Accuracy Results")
spam_plot



# #MNIST Plot
MNIST_loaded_data = np.load("/Users/dankim/Downloads/Cal Stuff/Cal Coursework F2020-Sp2025/Cal Berkeley Spring 2025/CS 189/Homework/hw1/data/mnist-data.npz")
m_training_data = MNIST_loaded_data["training_data"]
m_training_data = m_training_data.reshape(m_training_data.shape[0], -1)
m_training_labels = MNIST_loaded_data["training_labels"]
m_num_vals = 10000

pM_train_data, pM_train_labels, pM_validation_data, pM_validation_labels = data_partitioning(m_training_data, m_training_labels, m_num_vals)

#Train SVC Model for SVM
p_train_plt_data, p_validation_plt_data = individual_SVM_training(ms_training_size, pM_train_data, pM_train_labels, pM_validation_data, pM_validation_labels)

MNIST_plot = plot_classification_accuracy(ms_training_size, p_train_plt_data, p_validation_plt_data, "MNIST Classification Accuracy Results")
MNIST_plot
```

5 Hyperparameter Tuning

In the previous problem, by training SVM models with varying amounts of training examples, you learned parameters for a model that classifies the data. Many classifiers also have *hyperparameters* that you can tune to influence the parameters. In this problem, we'll determine good values for the regularization parameter C in the soft-margin SVM algorithm. The interpretation of this parameter, as well as the functioning of the soft-margin SVM will be covered in lecture. For now, consider C as a parameter of a black-box algorithm that we aim to optimize.

When we are trying to choose a hyperparameter value, we train the model repeatedly with different hyperparameters. We select the hyperparameter that gives the model the highest *validation* accuracy. Before generating predictions for the test set, the model should be retrained using all the labeled data (including the validation data) and the previously-determined hyperparameter. **The use of automatic hyperparameter optimization libraries is strictly prohibited.**

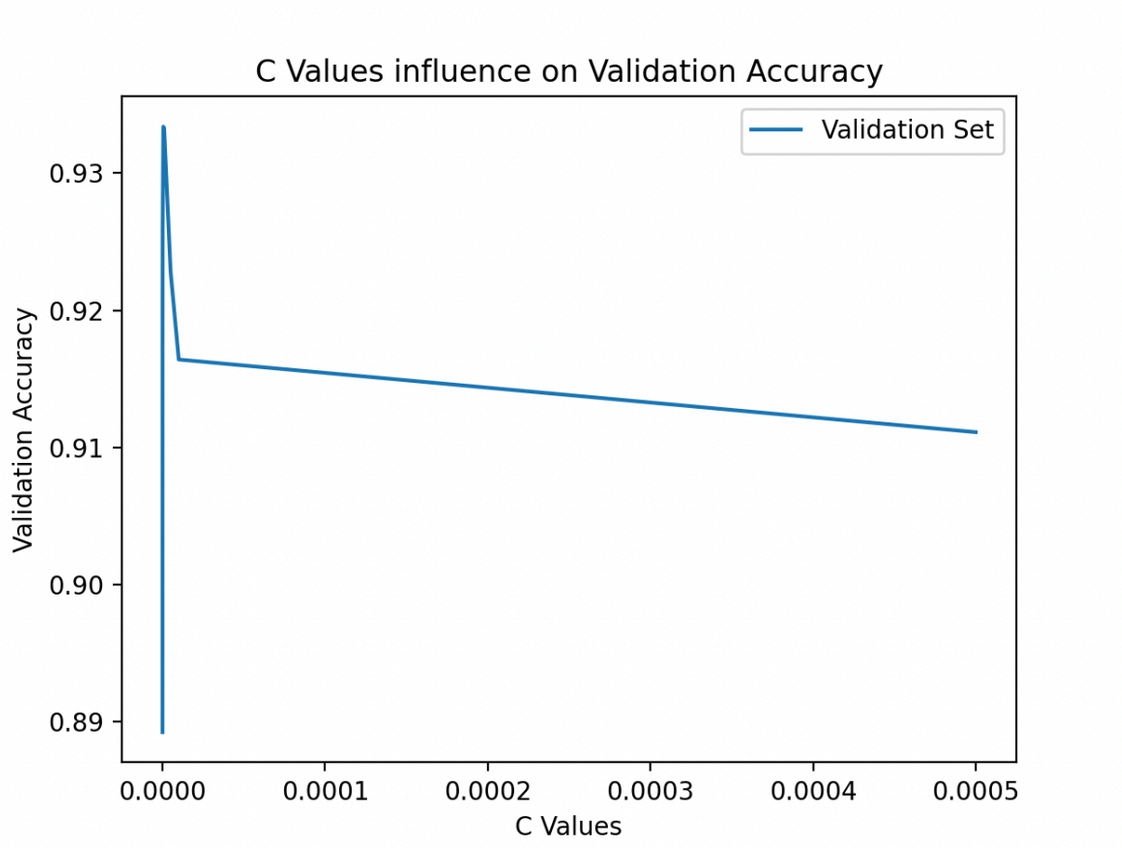
For the MNIST dataset, train a linear SVM model with various values of C to determine the best value for this hyperparameter. You should try different orders of magnitude of C values (e.g., 0.1, 1, 10, etc.). For performance reasons, you are required to train with at least 10,000 training examples. You can train on more if you like, but it is not required.

Deliverable: In your report, list at least 8 C values you tried, the corresponding accuracies, and the value corresponding to the highest validation accuracy. Reference any code you used to perform a hyperparameter sweep in the code appendix.

```
C Values
c_values = [1e-08, 5e-08, 1e-07, 5e-07, 1e-06, 5e-06, 1e-05, 5e-04]


C Values and Corresponding Accuracy
[[1e-08, np.float64(0.8892)], [5e-08, np.float64(0.9197)], [1e-07,
np.float64(0.9259)], [5e-07, np.float64(0.9334)], [1e-06,
np.float64(0.9333)], [5e-06, np.float64(0.9228)], [1e-05,
np.float64(0.9164)], [0.0005, np.float64(0.9111)]]


Best Combination: [5e-07, np.float64(0.9334)]
```


C Values influence on Validation Accuracy
Validation Set
0.93
0.92
0.91
0.90
0.89
Validation Accuracy
0.0000
0.0001
0.0002
0.0003
0.0004
0.0005
C Values

```python
# Question 5: Hyperparameter Tuning

c_values = [1e-08, 5e-08, 1e-07, 5e-07, 1e-06, 5e-06, 1e-05, 5e-04]

def plot_c_values_validation(c_values, validation_accuracy):
    plt.title("C Values influence on Validation Accuracy")
    plt.plot(c_values, validation_accuracy, label="Validation Set")
    plt.xlabel("C Values")
    plt.ylabel("Validation Accuracy")
    plt.legend()
    plt.show()

def c_val_model(x, y, c):
    svm_model = SVC(kernel="linear", C = c)
    svm_model.fit(x, y)
    return svm_model

def c_val_test(train_data, train_labels, validation_data, validation_labels, c_vals):
    val_acc = []
    res = []
    for C in c_vals:
        model = c_val_model(train_data[:10000], train_labels[:10000], C)
        predict = model.predict(validation_data)
        validation_acc = evaluation_metric(predict, validation_labels)
        val_acc.append(validation_acc)
        res.append([C, validation_acc])
    return val_acc, res

m_validation_accuracy, res = c_val_test(pM_train_data, pM_train_labels, pM_validation_data, pM_validation_labels, c_values)
plot_c_values_validation(c_values, m_validation_accuracy)
print(res)
```

6 K-Fold Cross-Validation

For smaller datasets (like the spam dataset), the validation set contains fewer examples, which means that our estimate of accuracy has high variance and thus might not be accurate. A way to combat this is to use *k-fold cross-validation*.

In k-fold cross-validation, the training data is shuffled and partitioned into k disjoint sets. Then the model is trained on $k-1$ sets and validated on the k^{th} set. This process is repeated k times with each set chosen as the validation set once. The cross-validation accuracy we report is the accuracy averaged over the k iterations. **The use of automatic cross-validation libraries is prohibited.**

For the spam dataset, use 5-fold cross-validation to find and report the best C value. As in problem 5, you should try different orders of magnitude of C. **Hint:** Effective cross-validation requires choosing from *random* partitions. This is best implemented by randomly shuffling your training examples and labels, and then partitioning them by their indices.

Deliverable: In your report, list at least 8 C values you tried, the corresponding accuracies, and the best C value. Reference any code you used to perform cross validation in the code appendix.

```
C Values:
[0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 10]

C Values and Corresponding Accuracy
[[1e-05, np.float64(0.7168048262864084)], [5e-05, np.float64(0.7183031802484408)...
np.float64(0.7183031802484408)], [0.0001, np.float64(0.7236968959466454)],
[0.0005, np.float64(0.7467719055270571)], [0.001,
np.float64(0.7566633723496996)], [0.005, np.float64(0.7836328472535034)],
[0.01, np.float64(0.7929219246878803)], [0.05,
np.float64(0.8145048551957205)], [0.1, np.float64(0.8267915369669426)], [0.5,
np.float64(0.8390746330870439)], [1, np.float64(0.8411699979606609)], [10,
np.float64(0.8453732774868172)]]

Best Combination:
[10, np.float64(0.8453732774868172)]
```

```python
#Question 6: K-Fold Cross-Validation

c_values_spam = [0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 10, 100]

def xy_start_end(k, size_val, random_index, train_data, train_labels, total_num):
    x_vals = []
    y_vals = []
    for i in range(k):
        start = i * size_val
        if i == k - 1:
            end = total_num
        else:
            end = (i+1) *  size_val
        x_vals.append(train_data[random_index[start:end]])
        y_vals.append(train_labels[random_index[start:end]])
    return x_vals, y_vals

def k_fold_cross_validation(k, train_data, train_labels, c_vals):
    total_num = len(train_data)
    random_index = np.random.permutation(total_num)
    size_val = total_num // k
    x_vals, y_vals = xy_start_end(k, size_val, random_index, train_data, train_labels, total_num)

    res = []
    vali_vals = []
    for C in c_vals:
        validation_values = []
        for i in range(k):
            x_val = x_vals[i]
            y_val = y_vals[i]
            x_train = np.concatenate(x_vals[:i] + x_vals[i+1:], axis = 0)
            assert x_train.shape[0] + x_val.shape[0] == total_num
            y_train = np.concatenate(y_vals[:i] + y_vals[i+1:], axis = 0)
            assert y_train.shape[0] + y_val.shape[0] == total_num
            k_fold_model = c_val_model(x_train[:10000], y_train[:10000], C)
            predict = k_fold_model.predict(x_val)
            validation_values.append(evaluation_metric(predict, y_val))
        res.append([C, np.mean(validation_values)])
        vali_vals = validation_values
    return res, vali_vals

k_res, k_validation_accuracy = k_fold_cross_validation(5, sM_train_data, sM_train_labels, c_values_spam)
print(k_res)
```

7 Kaggle

- MNIST Competition: https://www.kaggle.com/t/4bd3a8ef94dc4b3e88b57d45251742a6
- SPAM Competition: https://www.kaggle.com/t/dd35c1b3d87346458938b74689a3a5d6

Using an SVM model, generate predictions for the two test sets we provide and save those predictions to CSV files. The CSV file should have two columns–`Id` and `Category`–with a comma as a delimiter between them. The Id column should start at the integer 1 and end at the number of elements in the test set. The category label should be a dataset-dependent integer (one of $\{0, \ldots, 9\}$ for MNIST and one of $\{0, 1\}$ for spam. Be sure to use integer labels (not floating-point), and ensure that there are no spaces (not even after the commas). You can use the function below (copied from `save_csv`) to generate the CSV files:

```
def results_to_csv(y_test, file_name):
    y_test = y_test.astype(int)
    df = pd.DataFrame({'Category': y_test})
    df.index += 1
    df.to_csv(file_name, index_label='Id')
```

To check that your CSV files are formatted correctly, use `scripts/check.py` as a sanity check:

```
python check.py <competition name, eg. mnist> <submission csv file>
```

Once you have properly formatted CSV files with your predictions, upload your predictions to the Kaggle leaderboards and view the accuracy of your models. **Note that Kaggle only permits two submissions per leaderboard per day, so please start early!**

General comments about the Kaggle sections of homeworks: Most or all of the coding homeworks will include a Kaggle section. Whereas other parts of the homework might impose strict limits on what methods and libraries you are permitted to use, the Kaggle portions permit you to apply your creativity and find clever ways to improve your leaderboard performance. (Although extensive creativity is not generally necessary to get full points on an assignment, topping the Kaggle leaderboard gives your professor good material for letters of recommendation.)

The main restriction for Kaggle competitions is that you cannot use an entirely different learning technique. For example, this is an SVM homework, so you must use an SVM. (You are not permitted to use a neural network or a decision tree instead of, or in addition to, an SVM.) You are also absolutely not allowed to search for the labeled test data and submit that to Kaggle.

There are many other allowable ways to achieve higher positions on the Kaggle leaderboards. For example, you may use a nonlinear SVM kernel or add/remove features to/from the data. (For reasons we will learn later this semester, dropping features that have little or no predictive power will often improve your test performance as much as adding the right new features.) Spam is a particularly good dataset for playing with feature engineering. One easy way to perform better in the spam competetion is to add extra features with `featurize.py`. (We describe how you might do this below.) For this dataset, you could also look into using a bag-of-words model. For image data (i.e., MNIST), you could explore SIFT and HOG features.

Whatever creative ideas you apply, please explain what you did in your write-up. Cite any external sources you used to get ideas. If you have any questions about whether something is allowed or not, please ask on Ed Dicussion to avoid being penalized.

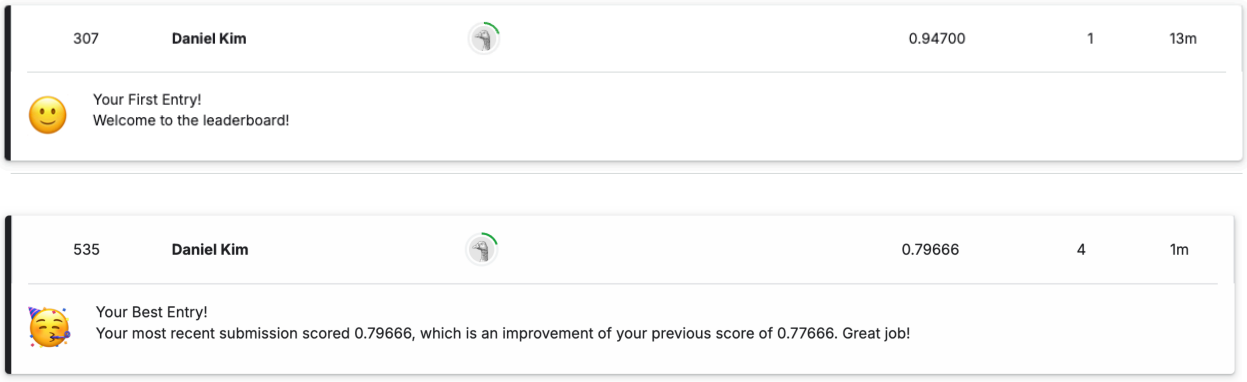
Deliverable: Your deliverable for this question has three parts. First, include a screenshot of your place on the Kaggle leaderboard for each of the datasets. (Be sure to include your Kaggle name, your score, and the time of your last submission. The time of your submission must be before your submission of the homework on Gradescope.) Second, include an explanation of what you tried, what worked, and what did not work to improve your accuracy. Finally, make sure to include all your code in the code appendix and provide a reference to it.

Modifying features for spam: The Python script `scripts/featurize.py` extracts features from the original emails in the spam dataset. The spam emails can be found in `data/spam/`, the ham emails can be found in `data/ham/`, and the emails for the test set can be found in `data/test/`. You are encouraged to look at the emails and try to think of features you think would be useful in classifying an email as spam or ham.

To add a new feature, modify `featurize.py`. You are free to change the structure of the code provided, but if you are following the given structure, you need to do two things:

- Define a function (e.g., `my_feature(text, freq)`) that computes the value of your feature for a given email. The argument `text` contains the raw text of the email, and `freq` is a dictionary containing the counts of each word in the email (or 0 if the word is not present). The value you return should be an integer or a float.
- Modify `generate_feature_vector` to append your feature to the feature vector.

Once you are done modifying `scripts/featurize.py`, re-generate the training and test data by running this script from within the `scripts` directory.



```python
# Question 7: Kaggle:

s_test_data = spam_loaded_data["test_data"]
s_test_data = s_test_data.reshape(s_test_data.shape[0], -1)
m_test_data = MNIST_loaded_data["test_data"]
m_test_data = m_test_data.reshape(m_test_data.shape[0], -1)

def results_to_csv(y_test, file_name):
    y_test = y_test.astype(int)
    df = pd.DataFrame({"Category": y_test})
    df.index += 1
    df.to_csv(file_name, index_label="Id")
    print(f"CSV saved as {file_name}")

def testing_Kaggle(train_data, train_labels, test_data, c_val, file_name):
    model = c_val_model(train_data, train_labels, c_val)
    test_label = model.predict(test_data)
    results_to_csv(test_label, file_name)

m_question7 = testing_Kaggle(m_training_data, m_training_labels, m_test_data, 5e-07, "MNIST_Test_Labels.csv")
s_question7 = testing_Kaggle(s_training_data, s_training_labels, s_test_data, 10, "spam_Test_Labels.csv")
```

To keep increasing the value of the validation accuracy for SPAM, I kept adding a number of features that I found to be extremely present in SPAM emails. For example, upon investigating, I found that China and money were words that were used a lot in SPAM emails. While words like Money were already incorporated, I added words like China and free into my features that helped train and elevate the validation score of my code. For MNIST, my code was able to discern the test data at a very high clip ~95%, and required only my first attempt to pass the test. Words that didn't help that I tried were like reservation, which made me rethink a lot of words that I chose for features. Additionally phrases like "you win" hurt the validation score and I didn't realize this until removing all phrases from the features list.