CS 189 Introduction to Machine Learning
Spring 2024 Jonathan Shewchuk

HW1

Due: Wednesday, January 24 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 flattened grayscale, 28×28 pixels. There are 10 possible labels for each image, namely, the digits 0–9.

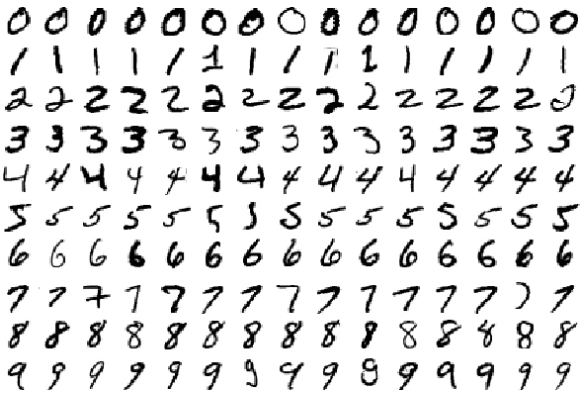


Figure 1: Examples from the MNIST dataset.

- **spam-data.npz** contains featurized spam data. The labels are 1 for spam and 0 for ham. The `data` folder includes the script **featurize.py** and the folders **spam**, **ham** (not spam), and **test** (unlabeled test data); you may modify **featurize.py** to generate new features for the spam data.

To check whether your Python environment is configured properly for this homework, ensure the following Python script executes without error. 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).

```
# This file is in scripts/load.py
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 ["mnist", "spam", "toy"]:
        data = np.load(f"../data/{data_name}-data.npz")
        print("\nloaded %s data!" % data_name)
        fields = "test_data", "training_data", "training_labels"
        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 : Lan Dinh

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. The primal hard-margin SVM optimization problem (which chooses the 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 $\|w\| = \sqrt{w \cdot w}$.

We can rewrite this optimization problem by using Lagrange multipliers to eliminate the constraints. (If you're curious to know what Lagrange multipliers are, the Wikipedia page is recommended, 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 Wikipedia 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:

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

Question 2

SVM : $\min_{w,\alpha} \|w\|^2$

$$s.t \quad y_i(X_i w + \alpha) \geq 1 \quad ; \forall i \in \{1, \dots, n\}$$

Dual $d^* = \max_{\lambda_i \geq 0} \min_{w,\alpha} \|\vec{w}\|^2 - \sum_{i=1}^{n} \lambda_i \left(y_i(X_i \vec{w} + \alpha) - 1\right) \quad (3)$

a) Show (3) can be rewritten as

$$\max_{\lambda_i \geq 0} \sum_{i=1}^{n} \lambda_i - \frac{1}{4} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j X_i X_j$$

$$s.t: \sum_{i=1}^{n} \lambda_i y_i = 0$$

- $f(\vec{w}, \alpha) = \|\vec{w}\|^2 - \sum_{i=1}^{n} \lambda_i \left(y_i(X_i \vec{w} + \alpha) - 1\right)$

$$\nabla_{\vec{w}} f(\vec{w}, \alpha) = 2\vec{w} - \sum_{i=1}^{n} \lambda_i y_i X_i = 0$$

$$\vec{w}^* = \frac{\sum_{i=1}^{n} \lambda_i y_i X_i}{2} \quad (\star)$$

$$\nabla_{\alpha} f(\vec{w}, \alpha) = \sum_{i=1}^{n} \lambda_i y_i = 0 \quad (\star\star)$$

Let $g(\lambda) = \min_{\vec{w},\alpha} f(\vec{w}, \alpha) = f(\vec{w}^*, \alpha)$

$$g(\lambda) = \|\vec{w}^*\|^2 - \sum_{i=1}^{n} \left[\lambda_i \left(y_i(X_i \vec{w}^* + \alpha) - \lambda_i\right)\right]$$

$$= \frac{1}{4}\left[\left(\sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j X_i X_j\right)\right] - \frac{1}{2}\left[\sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j X_i X_j\right)$$

$$- \underbrace{\sum_{i=1}^{n} \lambda_i y_i}_{=0 \text{ b/c } (\star\star)} \alpha + \sum_{i=1}^{n} \lambda_i$$

$$g(\lambda) = \sum_{i=1}^{n} \lambda_i - \frac{1}{4} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j X_i X_j \quad s.t \ \sum \lambda_i y_i = 0$$

Thus, (3) can be rewritten as

$$d^* = \max_{\lambda_i \geq 0} g(\lambda) = \max_{\lambda_i \geq 0} \sum_{i=1}^{n} \lambda_i - \frac{1}{4} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j X_i X_j$$

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

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

from a), $w^* = \frac{1}{2} \sum_{i=1}^{n} \lambda_i^* y_i X_i$

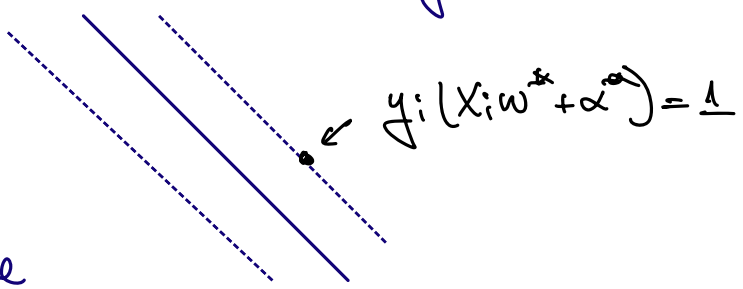
Plug-in (1), we have: $r(x) = \begin{cases} +1 & \text{if } \alpha^* + \frac{1}{2} \sum_{i=1}^{n} \lambda_i^* y_i X_i \cdot x \geq 0 \\ -1 & \text{otherwise} \end{cases}$

c) KKT conditions:

- Primal Feasibility: $y_i(X_i w^* + \alpha^*) - 1 \geq 0$
- Complementary Slackness: $\lambda_i^* [y_i (X_i w^* + \alpha^*) - 1] = 0$ (***) $\forall i \in \{1 \dots n\}$

when $\lambda_i^* > 0$, $y_i(X w^* + \alpha^*) - 1 = 0$

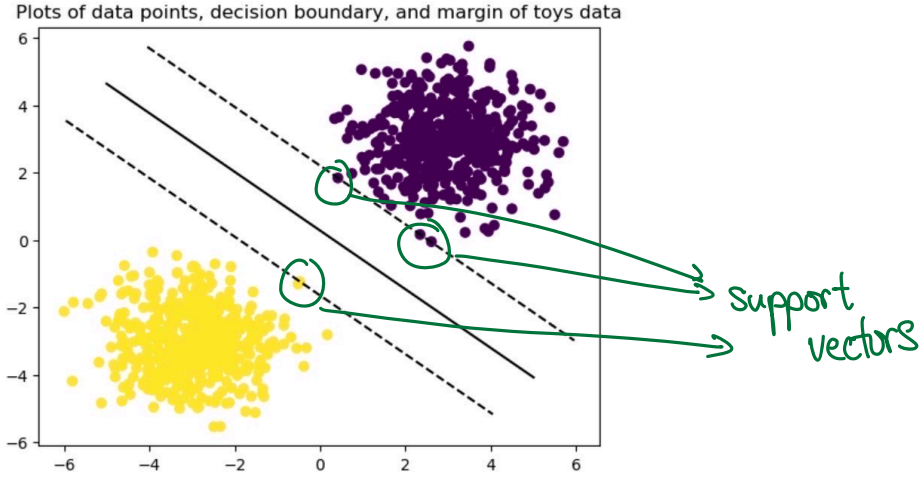
This means those points have to lie on the margin



d) When X_i is not lying on the margin (X_i is not a support vector),

$\Rightarrow y_i (X_i w^* + \alpha^*) - 1 \neq 0$. In order for (***) to hold, λ_i must be equal to 0. This translates to that we don't have λ_i^* to evaluate the decision rule.

e)



Question 3

a) Data Partitioning

[53]:
```
### Function to split data
def split_train_val(data, labels, num_val, num_train):
    ## take arguments:
    ## training data, label data, number of data in val set, and number of data in train set
    ## return an array of val_x, train_x, val_y, train_y

    shuffled_indices = np.random.permutation(num_train+num_val)

    val_indices = shuffled_indices[:num_val]
    train_indices = shuffled_indices[num_val:]

    val_x = data[val_indices, :].reshape(num_val, -1)
    train_x = data[train_indices, :].reshape(num_train, -1)

    val_y = labels[val_indices]
    train_y = labels[train_indices]

    return [val_x, train_x, val_y, train_y]
```

[54]:
```
## Set aside 10,000 training mnist data as a validation set
mnist_len = mnist_training_data.shape[0]
mnist_num_val = 10000
mnist_num_train = mnist_len - mnist_num_val

mnist_split = split_train_val(mnist_training_data, mnist_labels, mnist_num_val, mnist_num_train)

mnist_val_x = mnist_split[0]
mnist_train_x = mnist_split[1]
mnist_val_y = mnist_split[2]
mnist_train_y = mnist_split[3]
```

[55]:
```
## Set aside 20% of the training spam data as a validation set
spam_len = spam_training_data.shape[0]
spam_num_val = int(spam_len*0.2)
spam_num_train = spam_len - spam_num_val

spam_split = split_train_val(spam_training_data, spam_labels, spam_num_val, spam_num_train)

spam_val_x = spam_split[0]
spam_train_x = spam_split[1]
spam_val_y = spam_split[2]
spam_train_y = spam_split[3]
```

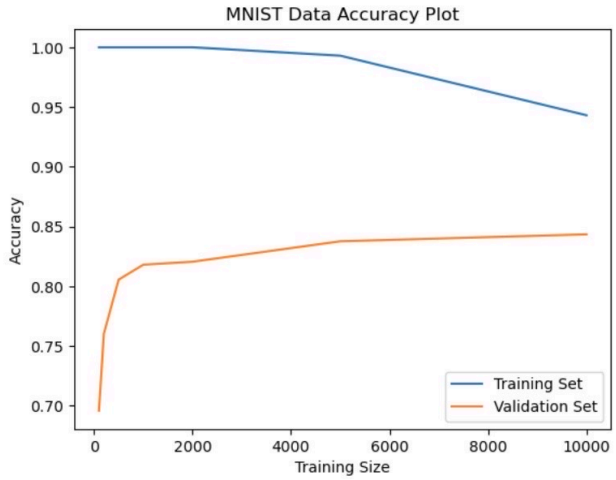
b) Evaluation Metric

3b. Evaluation metric

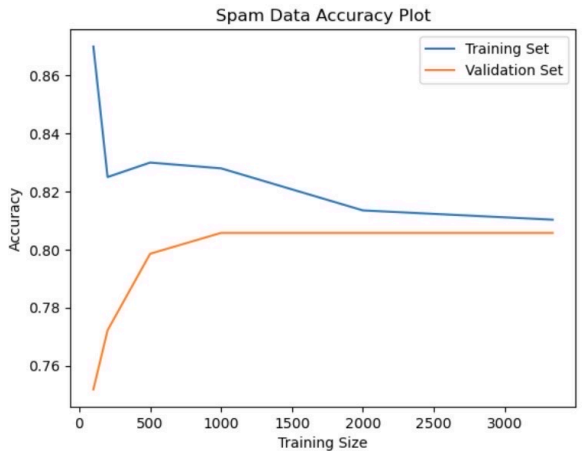
:
```
def classification_accuracy(true_y, predicted_y):
    assert len(true_y) == len(predicted_y)
    return np.sum(true_y == predicted_y)/len(true_y)
```

Question 4

(a)



b)



Question 5

Range of C: [1e-06, 1) ; step = 0.1

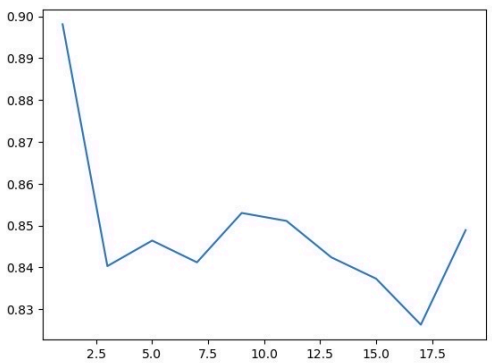
The corresponding validation accuracies is as followed:

```
C:  1e-06  has validation accuracy:  0.8981
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the numbe
  warnings.warn(
C:  0.100001  has validation accuracy:  0.8403
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the numbe
  warnings.warn(
C:  0.200001  has validation accuracy:  0.8464
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the numbe
  warnings.warn(
C:  0.300001  has validation accuracy:  0.8412
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the numbe
  warnings.warn(
C:  0.400001  has validation accuracy:  0.853
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the numbe
  warnings.warn(
C:  0.500001  has validation accuracy:  0.8511
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the numbe
  warnings.warn(
C:  0.6000010000000001  has validation accuracy:  0.8424
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the numbe
  warnings.warn(
C:  0.7000010000000001  has validation accuracy:  0.8373
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the numbe
  warnings.warn(
C:  0.8000010000000001  has validation accuracy:  0.8263
C:  0.900001  has validation accuracy:  0.8489
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the numbe
  warnings.warn(
```

[19]:
```
[0.8981, 0.8403, 0.8464, 0.8412, 0.853, 0.8511, 0.8424, 0.8373, 0.8263, 0.8489]
```

[45]:
```
# plt.plot(C_arr, val_accuracies)
```

[45]:
```
[<matplotlib.lines.Line2D at 0x7edbabc5b3a0>]
```



→ Best C = 0.000001 with validation accuracy = 0.8981

Question 6

Range of C : [1, 21) with step = 2

The corresponding accuracies are as below:

C = 19 has highest accuracy = 0.8175

```
  warnings.warn(
C  1  has mean validation accuracy:  0.8103545427131348
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
C  3  has mean validation accuracy:  0.8093961716854061
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
C  5  has mean validation accuracy:  0.8101161705366244
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
C  7  has mean validation accuracy:  0.8113137753270439
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
C  9  has mean validation accuracy:  0.8151507057826792
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
C  11  has mean validation accuracy:  0.8079564611783627
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
C  13  has mean validation accuracy:  0.8134717615129453
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
C  15  has mean validation accuracy:  0.8153902267407631
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
C  17  has mean validation accuracy:  0.8141914731687704
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
C  19  has mean validation accuracy:  0.8175473513404844
```

Question 7 Kaggle user name: Lanadore

- MNIST Kaggle score: 0.981
- SPAM Kaggle score: 0.993

- For mnist dataset, I first tried to find the optimal C value. However, my test accuracy never goes beyond 0.81. Then I seek for another way to engineer my feature and experiment HOG method (reference included in the appendix as well as in my code section). After using this method, I only use the default C value with my new engineered features. My validation and test accuracy improve significantly.

- For spam dataset, I first manually look for any more words/characters in the ham/spam files. However, this takes me too much time and inefficient to find optimal set of words/characters. Then I use a tool that can do this job for me more efficiently called TfidVectorizer. Using this tool to engineer my features as well as the optimal C-value I found in the above question (C = 11), my validation and test accuracy improve significantly.

Code Appendix

Question 2e) :

Ref: https://medium.com/geekculture/svm-classification-with-sklearn-svm-svc-how-to-plot-a-decision-boundary-with-margins-in-2d-space-7232cb3962c0

[47]:
```
### Load toy_data.npz
toy_data = np.load("../data/toy-data.npz")
toy_training_data = toy_data["training_data"]
toy_labels = toy_data["training_labels"]
```

[48]:
```
w = [-0.4525,-0.519]
b = 0.1471
```

[49]:
```
# Plot the decision boundary
x = np.linspace(-5, 5, 100)
y = -(w[0] * x +b) / w[1]
plt.plot(x, y, 'k')

# Plot the margins

## Unit vector of w
w_unit = w/(np.sqrt(np.sum(np.power(w,2))))
## Margin magnitude
margin_magnitude = 1/(np.sqrt(np.sum(np.power(w,2))))


## Margin lines
margin_upper = np.array(list((zip(x,y)))) + w_unit*margin_magnitude
margin_lower = np.array(list((zip(x,y)))) - w_unit*margin_magnitude


plt.scatter(toy_training_data[:, 0], toy_training_data[:, 1], c= toy_labels)


plt.title("Plots of data points, decision boundary, and margin of toys data")
plt.plot(margin_lower[:,0], margin_lower[:,1], 'k--')
plt.plot(margin_upper[:,0], margin_upper[:,1], 'k--')
```

Question 4 (Code Appendix)

a)

[57]:
```
def svm_model(train_x, train_y, C=1.0):
    ## return svm model
    mnist_svm_model= svm.LinearSVC(C=C)

    if train_x.ndim > 2:
        train_x = train_x.reshape(train_x.shape[0], -1)
    if train_y.ndim > 2:
        train_y = train_y.reshape(train_y.shape[0],-1)
    assert train_x.ndim <=2
    assert train_y.ndim <=2
    model = mnist_svm_model.fit(train_x, train_y)
    return model
```

[58]:
```
mnist_training_sizes = [100, 200, 500, 1000, 2000, 5000, 10000]
val_accuracies = []
train_accuracies = []

for s in mnist_training_sizes:
    sample_train_x = mnist_train_x[:s]
    sample_train_y = mnist_train_y[:s]
    mnist_svm = svm_model(sample_train_x, sample_train_y)

    val_predicted_y = mnist_svm.predict(mnist_val_x)
    val_accuracy = classification_accuracy(mnist_val_y, val_predicted_y)
    val_accuracies.append(val_accuracy)

    train_predicted_y = mnist_svm.predict(sample_train_x)
    train_accuracy = classification_accuracy(sample_train_y, train_predicted_y)
    train_accuracies.append(train_accuracy)
```

```
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
```

[59]:
```
plt.title("MNIST Data Accuracy Plot")
plt.xlabel("Training Size")
plt.ylabel("Accuracy")
plt.plot(mnist_training_sizes, train_accuracies, label ="Training Set")
plt.plot(mnist_training_sizes, val_accuracies, label ="Validation Set")
plt.legend()
```

b)

[60]:
```
spam_training_sizes = [100, 200, 500, 1000, 2000, spam_num_train]
val_accuracies = []
train_accuracies = []

for s in spam_training_sizes:
    sample_train_x = spam_train_x[:s]
    sample_train_y = spam_train_y[:s]
    spam_svm = svm_model(sample_train_x, sample_train_y)

    val_predicted_y = spam_svm.predict(spam_val_x)
    val_accuracy = classification_accuracy(spam_val_y, val_predicted_y)
    val_accuracies.append(val_accuracy)

    train_predicted_y = spam_svm.predict(sample_train_x)
    train_accuracy = classification_accuracy(sample_train_y, train_predicted_y)
    train_accuracies.append(train_accuracy)
```

```
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/srv/conda/lib/python3.9/site-packages/sklearn/svm/_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
```

[61]:
```
plt.title("Spam Data Accuracy Plot")
plt.xlabel("Training Size")
plt.ylabel("Accuracy")
plt.plot(spam_training_sizes, train_accuracies, label ="Training Set")
plt.plot(spam_training_sizes, val_accuracies, label ="Validation Set")
plt.legend()
```

Question 5 (code Appendix)

[18]:
```
## Function to tune hyperparamenter C

def hyper_tuning(x_val, x_train, y_val, y_train, C_arr):
    # return an array of corresponding accuracies on validation set
    accuracies = []
    for C in C_arr:
        model = svm_model(x_train, y_train, C)
        predicted_y = model.predict(x_val)
        accuracy = classification_accuracy(y_val, predicted_y)
        print("C: ", C, " has validation accuracy: ", accuracy)
        accuracies.append(accuracy)
    return accuracies
```

[19]:
```
# using 10 000 mnist images for training
mnist_num_train = 10000
C_arr = np.arange(0.000001, 1, 0.1)
val_accuracies = hyper_tuning(mnist_val_x, mnist_train_x[:mnist_num_train], mnist_val_y, mnist_train_y[:mnist_num_train], C_arr)
val_accuracies
```

Question 6 (code appendix)

[65]:
```
### Function implements k-fold-cross-validation

def k_fold_CV(features, labels, C_arr = np.array([1]), k=5):
    # shuffle indices
    assert features.shape[0] == labels.shape[0]
    data_len = labels.shape[0]
    shuffled_indices = np.random.permutation(data_len)

    # arrays of k partitions
    x_partitions = []
    y_partitions = []
    partitions_size = data_len // k

    #partition shuffled data into k folds
    for f in range(k):
        start_index = f*partitions_size
        if f == k-1:
            end_index = data_len
        else:
            end_index = (f+1)*partitions_size
        indices = shuffled_indices[start_index:end_index]
        if x.ndim > 2:
            x_partitions.append(features[indices,:].reshape(len(indices), -1))
        else:
            x_partitions.append(features[indices])
        y_partitions.append(labels[indices])
    #array of corresponding average val accuracies
    accuracies = []
    for C in C_arr:
        val_accuracies = []
        for f in range(k):
            x_val = x_partitions[f]
            y_val = y_partitions[f]

            x_train = np.concatenate(x_partitions[:f] + x_partitions[f+1:], axis= 0)
            y_train = np.concatenate(y_partitions[:f] + y_partitions[f+1:], axis= 0)

            model = svm_model(x_train, y_train, C)
            y_predicted = model.predict(x_val)
            val_accuracy = classification_accuracy(y_val, y_predicted)
            val_accuracies.append(val_accuracy)
        avg_val_accuracy = np.mean(val_accuracies)
        accuracies.append(avg_val_accuracy)
        print("C ", C, " has mean validation accuracy: ", avg_val_accuracy)

    return accuracies
```

[66]:
```
np.random.seed(42)
C_arr = np.arange(1, 21, 2)
# using 5-fold-CV on spam_training_data
spam_accuracies = k_fold_CV(spam_training_data, spam_labels, C_arr, 5)
spam_accuracies
```

Question 7 Code Appendix

MNIST Data

Feature Engineering

Reference: https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/

ChatGPT: question "How to implement HOG for SVM"

Set up

install opencv package by running the command ***pip install opencv-python***

install skimage by running the command ***pip3 install scikit-image***

[78]:
```
import cv2
import numpy as np

# Parameters for HOG
winSize = (28, 28)
blockSize = (14, 14)
blockStride = (7, 7)
cellSize = (7, 7)
nbins = 9
```

[79]:
```
def hog_feature(image_list):
    from skimage.feature import hog
    # HOG descriptor
    hog = cv2.HOGDescriptor(winSize, blockSize, blockStride, cellSize, nbins)

    hog_features_list = []
    image_list_reshaped = image_list.reshape(-1, 28, 28)
    for image in image_list_reshaped:
        # OpenCV's HOG Descriptor
        if image.shape != winSize or image.dtype != np.uint8:
            image = cv2.resize(image, winSize, interpolation=cv2.INTER_LINEAR)
            image = np.uint8(image * 255)

        hog_features = hog.compute(image)
        hog_features_list.append(hog_features)

    hog_features_reshape = np.array(hog_features_list).reshape(len(image_list), -1)
    return hog_features_reshape
```

[80]:
```
## Featurizing mnist data
mnist_train_x_hog = hog_feature(mnist_train_x)
mnist_val_x_hog = hog_feature(mnist_val_x)
```

Training

[81]:
```
mnist_model = svm_model(mnist_train_x_hog, mnist_train_y)
```

Evaluate Model

[82]:
```
mnist_y_predicted = mnist_model.predict(mnist_val_x_hog)
```

[83]:
```
mnist_val_accuracy = classification_accuracy(mnist_val_y, mnist_y_predicted)
mnist_val_accuracy
```

[83]:
```
0.9868
```

Spam data

Set up

[68]:
```
## Load spam dataset
spam_data = np.load("../data/spam-data.npz")
spam_training_data = spam_data["training_data"]
spam_labels = spam_data["training_labels"]
spam_test = spam_data['test_data']
```

[69]:
```
### Function to read spam and ham files
### Credit: function generate_design_matrix in /featurize.py

def read_file (filenames, category, storage):
    for filename in filenames:
        with open(filename, 'r', encoding='utf-8', errors='ignore') as f:
            try:
                text = f.read() # Read in text from file
            except Exception as e:
                # skip files we have trouble reading.
                continue
            text = text.replace('\r\n', ' ') # Remove newline character
            storage.append({'Content': text, 'Category' : category})
    return storage
```

[70]:
```
### Assign names for ham and spam file to read
### Credit: /featurize.py

from collections import defaultdict
import glob
import re
import scipy.io
import numpy as np
import pdb

NUM_TRAINING_EXAMPLES = 4172
NUM_TEST_EXAMPLES = 1000

BASE_DIR = '../data/'
SPAM_DIR = 'spam/'
HAM_DIR = 'ham/'
TEST_DIR = 'test/'

spam_filenames = glob.glob(BASE_DIR + SPAM_DIR + '*.txt')
ham_filenames = glob.glob(BASE_DIR + HAM_DIR + '*.txt')
test_filenames = [BASE_DIR + TEST_DIR + str(x) + '.txt' for x in range(NUM_TEST_EXAMPLES)]
```

[71]:
```
spam_training = []
spam_training = read_file(spam_filenames, 1, spam_training)
spam_training = read_file(ham_filenames, 0, spam_training)
```

[72]:
```
spam_training = pd.DataFrame(spam_training)
spam_training
```

Training

[73]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(spam_training['Content'])
tfidf_matrix.shape
```

[73]:
```
(4171, 43702)
```

[74]:
```
spam_val_x, spam_train_x, spam_val_y, spam_train_y = split_train_val(tfidf_matrix, spam_training['Category'], spam_num_val, spam_num_train)
```

[75]:
```
spam_model = svm_model(spam_train_x, spam_train_y, 11)
```

Evaluate model

[76]:
```
spam_y_predicted = spam_model.predict(spam_val_x)
```

[77]:
```
spam_accuracy = classification_accuracy(spam_val_y, spam_y_predicted)
spam_accuracy
```

[77]:
```
0.9856115107913669
```

This condition is called *complementary slackness*. Explain what this implies for points corresponding to $\lambda_i^* > 0$.

(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 evaluate 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, in your write-up, produce a plot of

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

In this plot, where are the support vectors?

Hint: You can use the following snippet, that plots the data points and decision boundary but not the margins.

```
plt.scatter(data[:, 0], data[:, 1], c=labels)

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

# Plot the margins
## TODO
```

(f) Assume the training points X_i and labels y_i are linearly separable. Using the original SVM formulation (not the dual) 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.

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, or "validation" dataset, to evaluate the performance of a model and ultimately select the best performing one, while using the rest of the data, or "training" dataset, to train models. In its simplest form, evaluating a trained model requires you to (i) have set aside a validation dataset, and (ii) selected 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. Write a function, taking as inputs the set of true labels y and the set of predicted labels $\hat{y}$, that computes the (unweighted) accuracy score s,

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

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

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

n is the total number of input observations, and for any $i \leq n$, 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!

4 Support Vector Machines: Coding

We will use linear Support Vector Machines (SVM) to classify our datasets. For images, 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.

Train a linear SVM on the spam and MNIST datasets. For each dataset, plot the accuracy 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.

You may use `sklearn` only for the SVM model. Everything else must be done without the use of `sklearn`.

(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. For the largest training set, you should expect validation accuracies between 70% and 90%. When you calculate the training accuracy, you only need to calculate on the subset of the data used to train the model, not necessarily the full training dataset.[2]

(b) For the **spam** dataset, use the provided word frequencies as features. In other words, each document is represented by a vector, where the i^{th} entry denotes the number of times word i (as specified in `featurize.py`) is found in that document. Train your model with the following numbers of training examples: 100, 200, 500, 1,000, 2,000, **ALL**. When you calculate the training accuracy, you only need to calculate on the subset of the data used to train the model, not necessarily the full training dataset.

For the largest training set, you should expect validation accuracies between 70% and 90%.

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. On MNIST, `LinearSVC` was faster on one member of Course Staff's laptop, though the exact results will likely depend on your computer, the parameters of the algorithm, and the data (number of data points vs number of features).

Deliverable: For this question, you should include two plots showing number of examples versus training and validation accuracy 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.

5 Hyperparameter Tuning

In the previous problem, 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 with the highest accuracy on the validation dataset. 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 prohibited for this part of the homework.

Deliverable: For the MNIST dataset, find the best C value. In your report, list at least 8 C values you tried, the corresponding accuracies, and the best C value. You should try a geometric sequence of C values (not an arithmetic sequence). As in the previous problem, 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. Again, reference any code you used to perform a hyperparameter sweep in the code appendix.

6 K-Fold Cross-Validation

For smaller datasets (e.g., the spam dataset), the validation set contains fewer examples, and our estimate of our accuracy might not be accurate—the estimate has high variance. 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.

Use of automatic cross-validation libraries is prohibited for this part of the homework.

Deliverable: For the spam dataset, use 5-fold cross-validation to find and report the best C value. In your report, list at least 8 C values you tried, the corresponding accuracies, and the best C value. Again, please include your code for cross validation or include a reference to its location in your code appendix.

Hint: Effective cross-validation requires choosing from **random** partitions. This is best implemented by randomly shuffling your training examples and labels, then partitioning them by their indices.

7 Kaggle

- MNIST Competition: https://www.kaggle.com/competitions/cs189-hw1-mnist-spring-2024
- SPAM Competition: https://www.kaggle.com/competitions/cs189-hw1-spam-spring-2024

With the best model you trained for each dataset, generate predictions for the 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 (for MNIST, one of $\{0, \ldots, 9\}$, and for spam, one of $\{0, 1\}$. **Be sure to use integer labels (not floating-point!) and no spaces (not even after the commas).** Upload your predictions to the Kaggle leaderboards (submission instructions are provided within each Kaggle competition). **In your write-up, include your Kaggle name as it displays on the leaderboard and your Kaggle score for each of the three datasets** .

General comments about the Kaggle sections of homeworks. Most or all of the coding homeworks will include a Kaggle section. Whereas other parts of a coding assignment might impose strict limits about what methods you're permitted to use, the Kaggle portions permit you to apply your creativity and find clever ways to improve your leaderboard performance. The main restriction 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) a support vector machine. (You are also not allowed to search for the labeled test data and submit that to Kaggle; that's outright cheating.)

For example, to achieve higher positions on the Kaggle leaderboards, you may optionally add more features or use a nonlinear SVM kernel. Spam is a particularly good dataset for playing with feature engineering; one easy way to perform better in spam/ham is to add extra features with `featurize.py` (see below). Other examples of things you might investigate include SIFT and HOG features for images, and a bag-of-words model for spam/ham. For reasons we'll 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. Although extensive creativity isn't generally necessary to get full points on an assignment, topping the Kaggle leaderboard gives your professor good material for letters of recommendation.

Whatever creative ideas you apply, please explain what you did in your write-up. Cite any external sources where you got ideas. If you have any questions about whether something is allowed or not, ask on Ed Discussion.

Remember to start early! Kaggle only permits two submissions per leaderboard per day. To help you format the submission so that Kaggle can interpret it correctly please use `scripts/check.py` to run a basic sanity check.

To check your submission csv,

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

Deliverable: Your deliverable for this question has three parts. First, submit to all the Kaggle competitions listed above, and include your Kaggle score in your write-up. Second, include an

explanation of what you tried, what worked, and what didn't to improve your accuracy. Finally, make sure to include all the code you used 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 (ie. not spam) 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, eg. `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; `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. For example:

```
feature.append(my_feature(text, freq))
```

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

```
python featurize.py
```