Sajal Ravish – CS 189 Homework 1

I worked with Aurelia Wang on this homework assignment.

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.

Sajal R.

(2) a) First, we have to show that Equation (3) is convex:

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

We know that $\|w\|^2$ is a norm and that all norms are convex. We also know that $\sum_{i=1}^{n} \lambda_i (y_i (X_i \cdot w + \alpha) - 1)$ is an affine function and that affine functions are convex. Since equation (3) is the difference between a convex function and an affine function, and subtracting an affine function from a convex one always results in a convex function, we can conclude that equation (3) is also a convex function. Using this information, we can do the following:

(continued on next page)

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

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

Take the partial derivative of this expression with respect to w:

$$\frac{\partial}{\partial w} \left(\max_{\lambda_i \geq 0} \min_{w, \alpha} \|w\|^2 - \sum_{i=1}^{n} \lambda_i y_i X_i w + \lambda_i y_i \alpha - \lambda_i \right)$$

$$= 2w - \sum_{i=1}^{n} \lambda_i y_i X_i \qquad \leftarrow \|w\|^2 = (\sqrt{w \cdot w})^2 = w^2$$

set our new expression equal to zero and solve for w to find the optimal value of w:

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

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

(continued on next page)

Now take the partial derivative of our original expression with respect to α:

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

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

Again, we find the optimal value of α by setting this expression equal to zero:

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

(continued on next page)

Now, we can substitute these new values back into the original expression and simplify:

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

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

$$= \max_{\lambda_i \geq 0} \min_{\alpha} \left(\frac{1}{2} \sum_{i=1}^{n} \lambda_i y_i x_i \right)^2 - \sum_{i=1}^{n} \frac{1}{2} \lambda_i y_i x_i \sum_{i=1}^{n} \lambda_i y_i x_i - \sum_{i=1}^{n} \lambda_i y_i \alpha + \sum_{i=1}^{n} \lambda_i$$

$$= \max_{\lambda_i \geq 0} \min_{\alpha} \frac{1}{4} \left(\sum_{i=1}^{n} \lambda_i y_i x_i \right)^2 - \frac{1}{2} \sum_{i=1}^{n} \lambda_i y_i x_i \sum_{i=1}^{n} \lambda_i y_i x_i - \alpha \sum_{i=1}^{n} \lambda_i y_i + \sum_{i=1}^{n} \lambda_i$$

(We proved this condition when finding $\frac{\partial}{\partial \alpha}$ of the original expression)

This equals zero under the condition $\sum_{i=1}^{n} \lambda_i y_i = 0$

$$= \max_{\lambda_i \geq 0} \frac{1}{4} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j x_i x_j - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i y_i x_i \lambda_j y_j x_j + \sum_{i=1}^{n} \lambda_i$$

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

$$= \max_{\lambda_i \geq 0} -\frac{1}{4} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j x_i x_j + \sum_{i=1}^{n} \lambda_i \quad \text{subject to} \quad \sum_{i=1}^{n} \lambda_i y_i = 0$$

$$= \max_{\lambda_i \geq 0} \sum_{i=1}^{n} \lambda_i - \frac{1}{4} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j x_i x_j \quad \text{subject to} \quad \sum_{i=1}^{n} \lambda_i y_i = 0$$

b) Decision rule for SVMs:

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

In part (a), we found that the optimal value for w, $w^* = \frac{1}{2} \sum_{i=1}^{n} \lambda_i y_i x_i$. Therefore, we can rewrite $r(x)$ as

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

Plugging in our value for w^*, we get:

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

assuming that values λ_i^* and α^* optimize the given equation (3)

c) We can rewrite the complementary slackness condition as:

$$\sum_{i=1}^{n} \lambda_i^* \left(y_i \left(X_i \cdot w^* + \alpha^* \right) - 1 \right) = 0$$

For points corresponding to $\lambda_i^* > 0$, this condition suggests that $y_i (x_i \cdot w^* + \alpha^*) - 1 = 0$ in order for the entire expression to evaluate to zero.

Therefore, $y_i (x_i \cdot w^* + \alpha^*) = 1$

(I got the following info from a public Ed discussion.)

The hyperplane $x_i \cdot w^* + \alpha^*$ is the decision boundary of the SVM (because points on either side of it are classified as different classes)

y_i is the ground-truth label of a data point and can either be +1 (indicating that x_i is "in class") or -1 (indicating that x_i is "not in class").

(continued on next page)

(I got the following information from the Karush-Kuhn-Tucker conditions Wikipedia page.)

Furthermore:
- λ_i is the Lagrange multiplier associated with the i^{th} data point
- X_i is the feature vector of the i^{th} data point
- w^* is the optimal weight vector for the SVM
- α^* is the optimal intercept for the SVM

① In the case where $\lambda_i = 0$: X_i is not a support vector

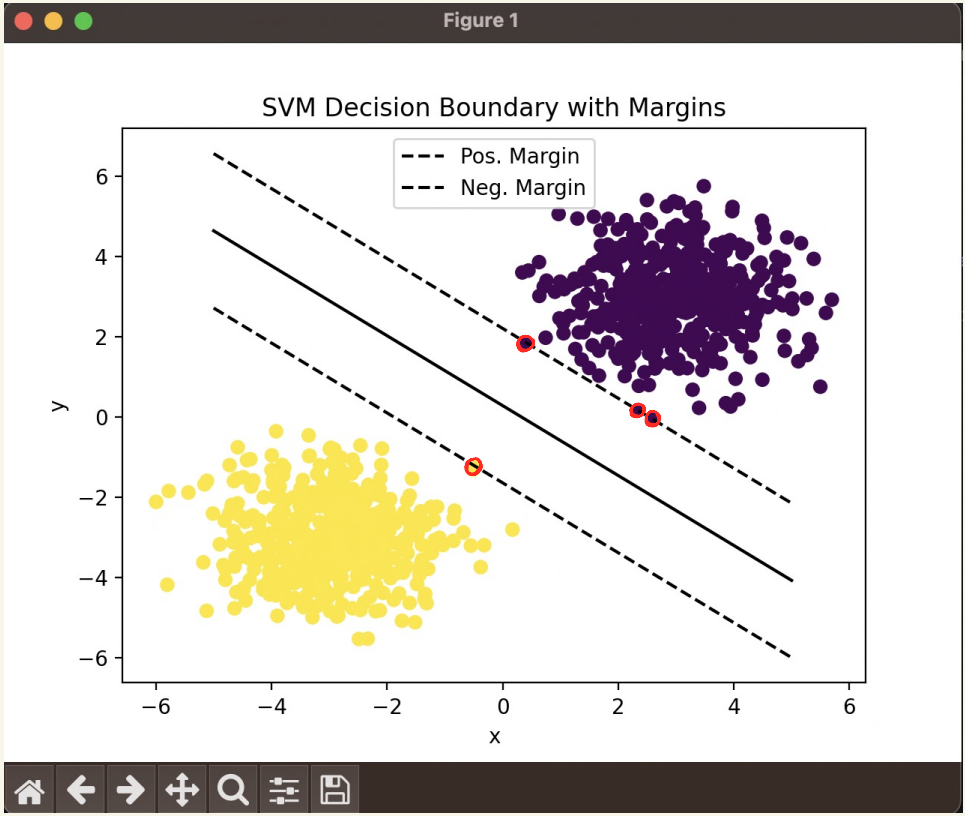
The point X_i is not on the margin (the separation between the decision boundary and the nearest data points from either class, which we aim to maximize), so its contribution to the objective function is zero. In other words, the Lagrange multiplier for the i^{th} data point doesn't affect optimization

② In the case where $\lambda_i > 0$: X_i is a support vector

The point X_i lies exactly on the margin, so its contribution to the objective function is exactly 1 because it helps determine the position of the decision boundary. Therefore, it must satisfy the condition $y_i(X_i \cdot w^* + \alpha^*) = 1$.

d) The decision rule in a SVM is the criteria used to classify new data points into different classes. It's based on only the support vectors (training points with nonzero Lagrange multipliers) because the decision boundary of an SVM is positioned in a way that maximizes the margin while still classifying vectors on the margin (which are the support vectors) correctly. Points that lie outside of the margin (which are typically the majority of training points in a data set) are therefore not relevant to determining the position of the hyperplane and therefore not needed to evaluate the decision rule. Furthermore, since the support vectors are frequently only a small minority of the total training points and the decision rule of an SVM is thereby influenced by only a small fraction of data, this makes SVMs computationally faster and very memory efficient. Since support vectors are the most challenging points to classify, this can also lead to a better decision rule.

e) My plot:



In this plot, the support vectors are the training points that lie on the margin (aka the "Pos. Margin" and "Neg. Margin" dotted lines). I have circled the support vectors in Figure 1 in the color red.

(continued on next page)

My code:

scripts > problem-2e.py > ...

```python
import numpy as np
import matplotlib.pyplot as plt

# Load toy data
toy_data = np.load(f"../data/toy-data.npz")
data, labels = toy_data["training_data"], toy_data["training_labels"]

# Define our variables (given in problem statement)
w1, w2 = -0.4528, -0.5190
w = np.array([w1, w2])
b = 0.1471

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
margin1 = (1 - w[0] * x - b) / w[1]
margin2 = (-1 - w[0] * x - b) / w[1]

plt.plot(x, margin1, "k--", label="Pos. Margin")
plt.plot(x, margin2, "k--", label="Neg. Margin")

plt.title('SVM Decision Boundary with Margins')
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```

f) Claim: For a given SVM, there is at least one support vector for each class (+1 and −1)

Proof by contradiction:

Let's assume that one of the classes has no support vectors. We want to show that, if there are no support vectors for one class, then we could decrease $\|w\|^2$ to get a more optimal solution. Therefore, our SVM parameters are not the solution to the optimization problem defined in the given equation (2).

Equation (2):

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

We can assume that the hyperplane separating the two classes in this hyperplane has parameters w and α that maximize the margin while still classifying all training points x_i with the correct labels y_i.

(continued on next page)

Let's assume that there are no support vectors for the class $+1$:

This implies that for all x_i with a corresponding $y_i = +1$ for all $i \in \{1, \ldots, n\}$, $y_i(x_i \cdot w + \alpha) > 1$. (In other words, $y_i(x_i \cdot w + \alpha) \neq 1$ because if the expression did equal to 1, x_i would be a support vector.) We can plug in the value $y_i = 1$ to get the new condition $x_i \cdot w + \alpha > 1$ (2)

Then, we can construct a new weight vector and corresponding bias such that they still meet condition (2):

$$w' = \frac{w}{1 + \frac{\varepsilon}{2}}$$

ε is a small, positive number, so w' is a slightly scaled-down version of w, making it a support vector. We can find α' given that w' is a support vector:

$y_i = 1$

$$y_i(x_i \cdot w' + \alpha') = 1$$

$$x_i\left(\frac{w}{1 + \frac{\varepsilon}{2}}\right) + \alpha' = 1$$

$$x_i \cdot w + \alpha'\left(1 + \frac{\varepsilon}{2}\right) = 1 + \frac{\varepsilon}{2}$$

$$\alpha'\left(1 + \frac{\varepsilon}{2}\right) = 1 + \frac{\varepsilon}{2} - x_i \cdot w$$

(continued on next pg)

$$\alpha' = \frac{(1+\frac{\varepsilon}{2})}{(1+\frac{\varepsilon}{2})} - \frac{x_i \cdot w}{(1+\frac{\varepsilon}{2})}$$

$$\alpha' = 1 - \frac{x_i \cdot w}{1+\frac{\varepsilon}{2}}$$

We can make the assumption that:

when $y_i = -1$

$x_i \cdot w + \alpha \leq -1$

when $y_i = +1$

$x_i \cdot w + \alpha \geq 1 + \varepsilon$

So we can plug in $x_i \cdot w = 1 + \varepsilon - \alpha$ for the $y_i = +1$ case:

$$\alpha' = 1 - \frac{1+\varepsilon-\alpha}{1+\frac{\varepsilon}{2}} = \frac{1+\frac{\varepsilon}{2} - 1 - \varepsilon + \alpha}{1+\frac{\varepsilon}{2}}$$

$$= \frac{-\frac{\varepsilon}{2} + \alpha}{1+\frac{\varepsilon}{2}} = \frac{2\alpha - \varepsilon}{\varepsilon + 2} \Rightarrow \alpha' = \frac{2\alpha - \varepsilon}{\varepsilon + 2}$$

Now we can prove that our new values of w' and α' satisfy our original constraint:

(continued on next page)

Our original constraint: $y_i(x_i \cdot w' + \alpha') \geq 1$

$$y_i = +1$$
$$w' = \frac{w}{1+\frac{\varepsilon}{2}}$$
$$\alpha' = \frac{2\alpha - \varepsilon}{\varepsilon + 2}$$

$$y_i(x_i \cdot w' + \alpha') = x_i\left(\frac{w}{1+\frac{\varepsilon}{2}}\right) + \left(\frac{\alpha - \frac{\varepsilon}{2}}{1+\frac{\varepsilon}{2}}\right)$$

$$= \frac{x_i w + \alpha - \frac{\varepsilon}{2}}{1+\frac{\varepsilon}{2}} \geq 1$$

Condition met because $\varepsilon > 0$, so $1 + \varepsilon > 1$ and $y_i(x_i w + \alpha) > 1$ where $y_i = +1$

$$x_i w + \alpha - \frac{\varepsilon}{2} \geq 1 + \frac{\varepsilon}{2}$$

$$x_i w + \alpha \geq 1 + \varepsilon > 1$$

$$y_i = -1$$
$$w' = \frac{w}{1+\frac{\varepsilon}{2}}$$
$$\alpha' = \frac{2\alpha - \varepsilon}{\varepsilon + 2}$$

$$y_i(x_i \cdot w' + \alpha') = -\left(x_i\left(\frac{w}{1+\frac{\varepsilon}{2}}\right) + \left(\frac{\alpha - \frac{\varepsilon}{2}}{1+\frac{\varepsilon}{2}}\right)\right)$$

$$= \frac{-x_i w - \alpha + \frac{\varepsilon}{2}}{1+\frac{\varepsilon}{2}} \geq 1$$

Condition met because $y_i = -1$, so $y_i(x_i w + \alpha) \geq 1$

$$-x_i w - \alpha + \frac{\varepsilon}{2} \geq 1 + \frac{\varepsilon}{2}$$

$$-x_i w - \alpha \geq 1$$

$$-(x_i w + \alpha) \geq 1$$

(continued on next page)

Now that we showed that w' and α' are valid and meet our given conditions / constraints (subject to $y_i(x_i \cdot w + \alpha) \geq 1, \forall i \in \{1, \ldots, n\}$ part of equation (2) is our constraint here), we can show that w' and α' actually offer a more optimal solution than w and α:

$$\|w'\|^2 < \|w\|^2$$

$$\left\| \frac{w}{1+\frac{\varepsilon}{2}} \right\|^2 < \|w\|^2$$

This is intuitively true because $\varepsilon > 0$, so $1 + \frac{\varepsilon}{2} > 1$ and dividing w by a number greater than 1 will make its magnitude smaller.

This proves that we found a more optimal solution because we are optimizing for the minimum value of $\|w\|^2$ in equation (2), and $\|w'\|^2 < \|w\|^2$. Herein lies the contradiction: our choice of w' and α' yields a larger margin (meaning, it's wider and further away from the decision boundary), which would not be possible if w and α were optimal.

(continued on next page)

Therefore, we can conclude that there must exist some x_i for which $x_i \cdot w + \alpha = 1$, which would be the support vector for the class labeled $y_i = +1$. We can apply a symmetric proof to the class labeled $y_i = -1$, which proves our original claim that, for a given SVM for a linearly separable dataset, there is at least one support vector for each class ($+1$ and -1).

Problem 3 Deliverables:

a) Data partitioning code:

Note: The split_ratio parameter of the partition function is what percent/number of training points we want to set aside for the validation set.

scripts > partition.py > ...

```python
################################################################
# QUESTION 3: data partitioning and evaluation metrics
import numpy as np

# data_name can be "mnist", "spam", or "toy"
def load_data(data_name):
    data = np.load(f"../data/{data_name}-data.npz")
    print("Loaded %s data!" % data_name)
    return data["training_data"], data["training_labels"]


def partition(data, labels, split_ratio):
    labels = labels.reshape((-1, 1))
    if data.ndim > 2:
        data = data.reshape((data.shape[0], -1))
    stacked_data = np.hstack((data, labels))

    if split_ratio <= 1:
        split_size = int(len(stacked_data) * split_ratio)
    else:
        split_size = split_ratio

    # randomly shuffle the data
    np.random.shuffle(stacked_data)

    # partition data
    training_data = stacked_data[split_size:]
    validation_data = stacked_data[:split_size]

    # separate features and labels for training and validation sets
    training_features = training_data[:, :-1]
    training_label = training_data[:, -1]

    validation_features = validation_data[:, :-1]
    validation_label = validation_data[:, -1]

    return training_features, training_label, validation_features, validation_label
```

```python


# Load MNIST dataset
mnist_data, mnist_labels = load_data("mnist")

# Partioning MNIST dataset (set aside )
mnist_training_features, mnist_training_label, mnist_validation_features, mnist_validation_label = partition(mnist_data, mnist_labels, 10000)

# Load spam dataset
spam_data, spam_labels = load_data("spam")

# Partioning spam dataset
spam_training_features, spam_training_label, spam_validation_features, spam_validation_label = partition(spam_data, spam_labels, 0.2)

print(f"\nMNIST Training Features: {mnist_training_features.shape}")
print(f"MNIST Training Labels: {mnist_training_label.shape}")
print(f"MNIST Validation Features: {mnist_validation_features.shape}")
print(f"MNIST Validation Labels: {mnist_validation_label.shape}")
print(f"\nSpam Training Features: {spam_training_features.shape}")
print(f"Spam Training Labels: {spam_training_label.shape}")
print(f"Spam Validation Features: {spam_validation_features.shape}")
print(f"Spam Validation Labels: {spam_validation_label.shape}")

```

b) Evaluation metric code:

```python

def accuracy_eval(true_labels, predicted_labels):
    if len(true_labels) != len(predicted_labels):
        raise ValueError("Inputs must have the same length!")
    total_labels = len(true_labels)

    # Calculate accuracy score
    correct_predictions = 0
    for i in range(total_labels):
        if true_labels[i] == predicted_labels[i]:
            correct_predictions += 1
    score = correct_predictions / total_labels

    return score

# Testing accuracy_eval:
true_labels = [1, 0, 2, 3, 3]
predicted_labels = [1, 0, 1, 0, 1]
score = accuracy_eval(true_labels, predicted_labels)
print(f"Accuracy: {score}")

```

External sources I used to solve this problem:

- https://numpy.org/doc/stable/reference/generated/numpy.reshape.html
- https://numpy.org/doc/stable/reference/random/generated/numpy.random.shuffle.html
- https://numpy.org/doc/stable/reference/generated/numpy.hstack.html

My code for this problem is located in the Code Appendix under the comment “QUESTION 3: data partitioning and evaluation metrics.”

Problem 4 Deliverables:

a) **MNIST dataset and plot:**

Training with 100 examples
Training accuracy: 1.0
Validation Accuracy: 0.79

Training with 200 examples
Training accuracy: 1.0
Validation Accuracy: 0.805

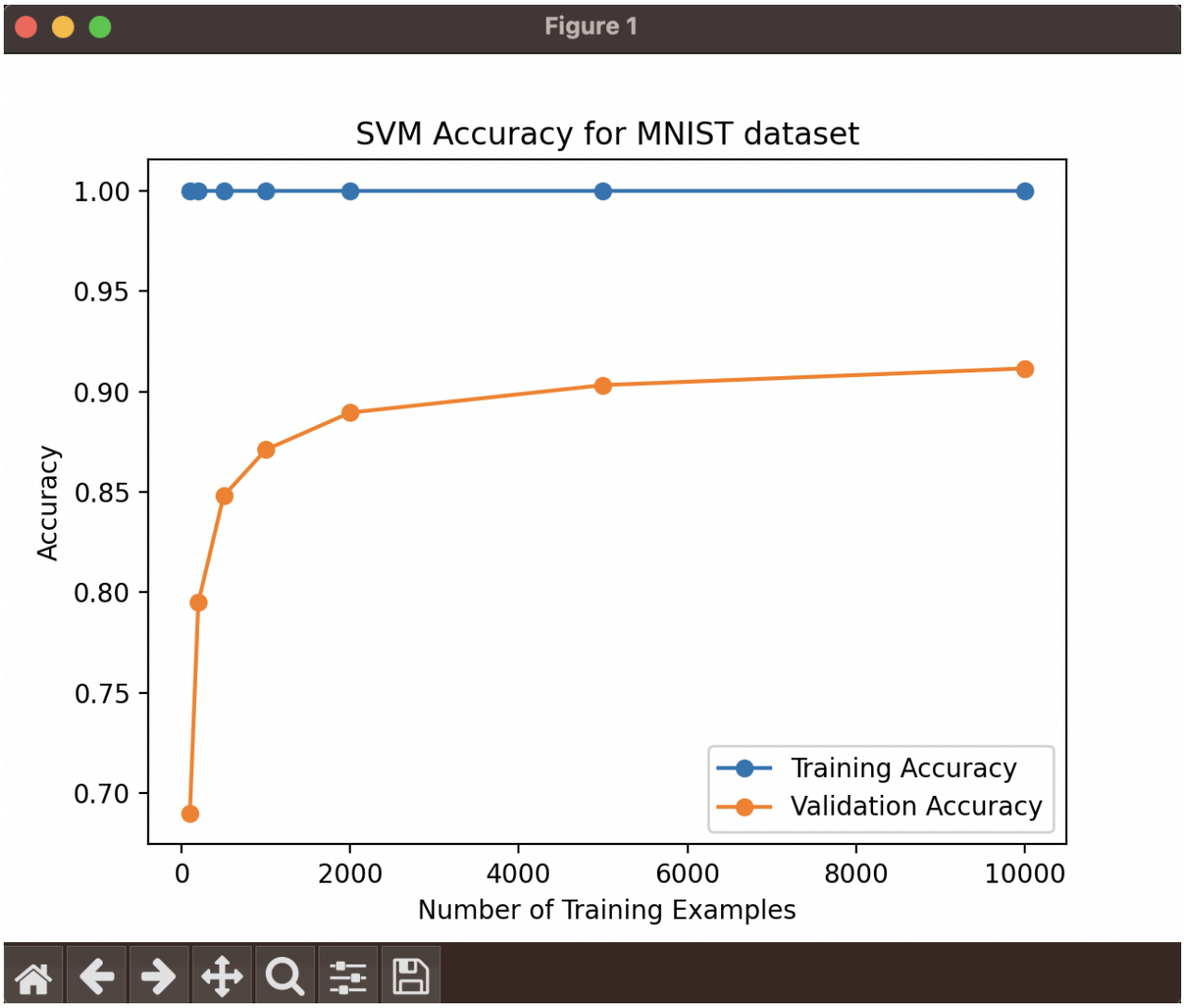
Training with 500 examples
Training accuracy: 1.0
Validation Accuracy: 0.88

Training with 1000 examples
Training accuracy: 1.0
Validation Accuracy: 0.875

Training with 2000 examples
Training accuracy: 1.0
Validation Accuracy: 0.898

Training with 5000 examples
Training accuracy: 1.0
Validation Accuracy: 0.9096

Training with 10000 examples
Training accuracy: 1.0
Validation Accuracy: 0.9104



b) Spam dataset and plot:

Training with 100 examples
Training accuracy: 0.92
Validation Accuracy: 0.72

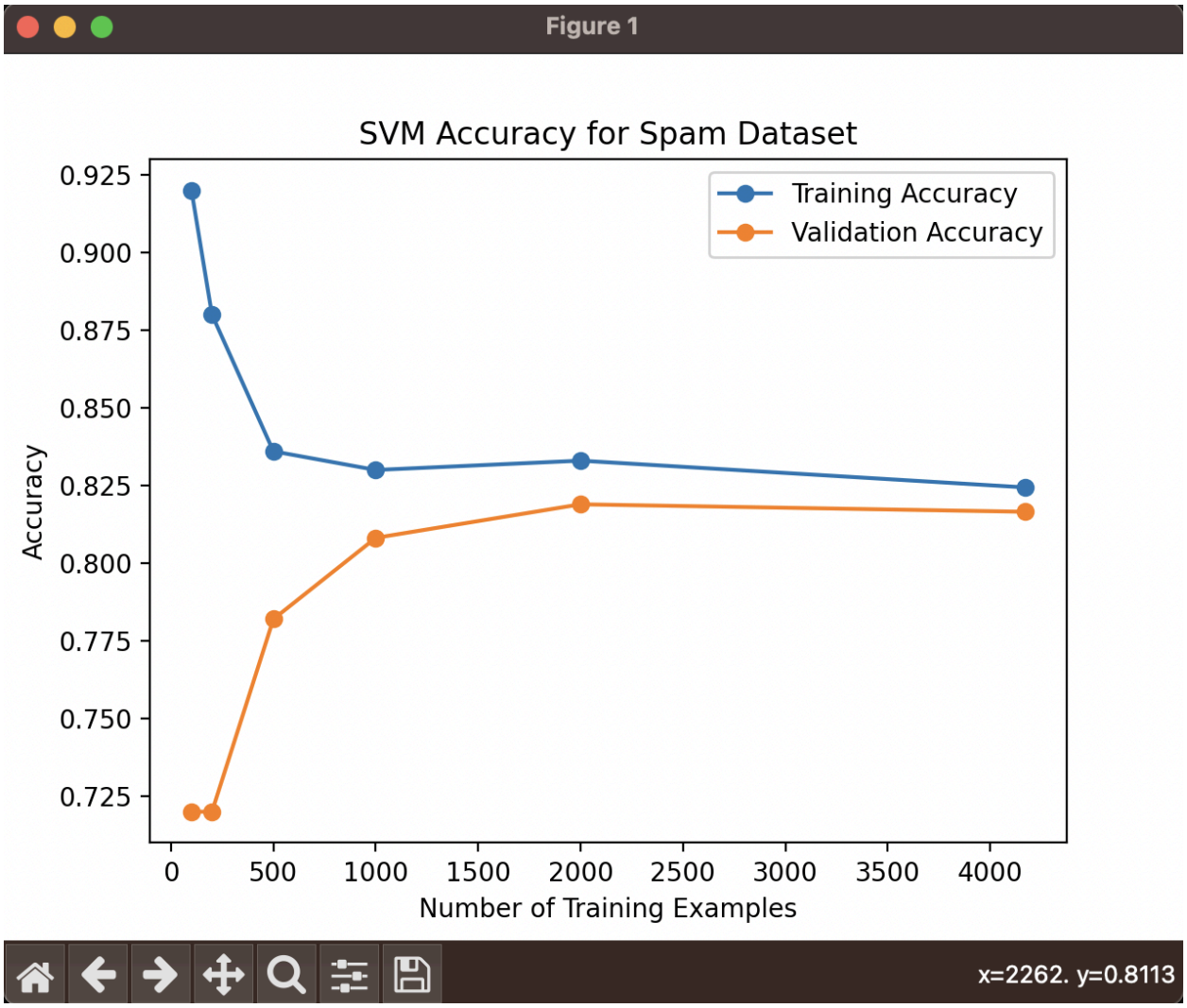
Training with 200 examples
Training accuracy: 0.88
Validation Accuracy: 0.72

Training with 500 examples
Training accuracy: 0.836
Validation Accuracy: 0.782

Training with 1000 examples
Training accuracy: 0.83
Validation Accuracy: 0.8081534772182254

Training with 2000 examples
Training accuracy: 0.833
Validation Accuracy: 0.8189448441247003

Training with 4171 examples
Training accuracy: 0.8243931675157327
Validation Accuracy: 0.8165467625899281



External sources I used to solve this problem:

- https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
- https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html

My code for this problem is located in the Code Appendix under the comment “QUESTION 4: support vector machines.”

Problem 5 Deliverables:

The values 15 I'm testing are: 1e-10, 1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 0.01, 0.1, 1, 10, 100, 1000, 10000.

Training with 1e-10 C-value
Training accuracy: 0.1143
Validation Accuracy: 0.112

Training with 1e-09 C-value
Training accuracy: 0.6713
Validation Accuracy: 0.6738

Training with 1e-08 C-value
Training accuracy: 0.8945
Validation Accuracy: 0.8906

Training with 1e-07 C-value
Training accuracy: 0.9335
Validation Accuracy: 0.9226

Training with 1e-06 C-value
Training accuracy: 0.9637
Validation Accuracy: 0.9285

Training with 1e-05 C-value
Training accuracy: 0.9922
Validation Accuracy: 0.9152

Training with 0.0001 C-value
Training accuracy: 0.9998
Validation Accuracy: 0.9089

Training with 0.001 C-value
Training accuracy: 1.0
Validation Accuracy: 0.9081

Training with 0.01 C-value
Training accuracy: 1.0
Validation Accuracy: 0.9081

Training with 0.1 C-value
Training accuracy: 1.0
Validation Accuracy: 0.9081

Training with 1 C-value
Training accuracy: 1.0
Validation Accuracy: 0.9081

Training with 10 C-value
Training accuracy: 1.0
Validation Accuracy: 0.9081

Training with 100 C-value
Training accuracy: 1.0
Validation Accuracy: 0.9081

Training with 1000 C-value
Training accuracy: 1.0
Validation Accuracy: 0.9081

Best C value: 0.0001
Best Validation Accuracy: 0.95435

My code for this problem is located in the Code Appendix under the comment “QUESTION 5: hyperparameter tuning.”

Problem 6 Deliverables:

The values 14 I'm testing are: 1e-10, 1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 0.01, 0.1, 1, 10, 100, 500.

C = 1e-10
Cross-Validation Accuracy = 0.7096243794270529

C = 1e-09
Cross-Validation Accuracy = 0.7096243794270529

C = 1e-08
Cross-Validation Accuracy = 0.7096243794270529

C = 1e-07
Cross-Validation Accuracy = 0.7096243794270529

C = 1e-06
Cross-Validation Accuracy = 0.7096243794270529

C = 1e-05
Cross-Validation Accuracy = 0.7096243794270529

C = 0.0001
Cross-Validation Accuracy = 0.7150167431254433

C = 0.001
Cross-Validation Accuracy = 0.7446843943297813

C = 0.01
Cross-Validation Accuracy = 0.7812418640978912

C = 0.1
Cross-Validation Accuracy = 0.8094107138047744

C = 1
Cross-Validation Accuracy = 0.8172023269802224

C = 10
Cross-Validation Accuracy = 0.8198987332680965

C = 500
Cross-Validation Accuracy = 0.8225955884333281

Best C value: 500
Best Cross-Validation Accuracy: 0.8225955884333281

External sources I used to solve this problem:

- https://medium.com/@avijit.bhattacharjee1996/implementing-k-fold-cross-validation-from-scratch-in-python-ae413b41c80d
- https://machinelearningmastery.com/k-fold-cross-validation/
- https://numpy.org/doc/stable/reference/generated/numpy.arange.html
- https://numpy.org/doc/stable/reference/generated/numpy.array_split.html

My code for this problem is located in the Code Appendix under the comment “QUESTION 6: k-fold cross-validation.”

Problem 7 Deliverables:

a) Kaggle score:
My Kaggle username: “Sajal Ravish”
MNIST dataset Kaggle score: 0.942
Spam dataset Kaggle score: 0.833

b) Improving model accuracy:
For the Spam dataset, I tried to improve my accuracy through feature engineering. I added 8 extra features to feature.py to search for 8 new words that I thought would be commonly found in spam emails. The 8 words were as follows: “please,” “improve,” “dear,” “content,” “mime,” “body,” “html,” and “offer.” I found these words through a combination of reading through the spam emails (found in the .txt files located in the data/spam folder) and observing common trends, as well as Googling “common words found in spam emails.” I tried using other words like “billion,” “cash,” and “click,” but they failed to improve the accuracy of my model. I also improved the efficiency of my model by using LinearSVC instead of SVC(kernel=’linear’) here (whereas I used SVC(kernel=’linear’) everywhere else in my code).

My accuracy score for the MNIST dataset was above the threshold on my first try, so I did not try to improve its accuracy.

My code for this problem is located in the Code Appendix under the comment “QUESTION 7: predictions for Kaggle.”

Code Appendix:

```
#####################################################################
# QUESTION 3: data partitioning and evaluation metrics
import numpy as np

# data_name can be "mnist", "spam", or "toy"
def load_data(data_name):
   data = np.load(f"../data/{data_name}-data.npz")
   print("Loaded %s data!" % data_name)
   return data["training_data"], data["training_labels"]


def partition(data, labels, split_ratio):
   # The split_ratio is what percent/number of training points we want to set aside
for the validation set

   # reshape and stack data and labels
   labels = labels.reshape((-1, 1))
   if data.ndim > 2:
       data = data.reshape((data.shape[0], -1))
   stacked_data = np.hstack((data, labels))

   # determine split_size
   if split_ratio <= 1:
       split_size = int(len(stacked_data) * split_ratio)
   else:
       split_size = split_ratio

   # randomly shuffle the data
   np.random.shuffle(stacked_data)

   # partition data
   training_data = stacked_data[split_size:]
   validation_data = stacked_data[:split_size]

   # separate features and labels for training and validation sets
   training_features = training_data[:, :-1]
   training_label = training_data[:, -1]

   validation_features = validation_data[:, :-1]
   validation_label = validation_data[:, -1]
```

```
    return training_features, training_label, validation_features, validation_label


# Load MNIST dataset
mnist_data, mnist_labels = load_data("mnist")

# Partitioning MNIST dataset (set aside )
mnist_training_features, mnist_training_label, mnist_validation_features,
mnist_validation_label = partition(mnist_data, mnist_labels, 10000)

# Load spam dataset
spam_data, spam_labels = load_data("spam")

# Partitioning spam dataset
spam_training_features, spam_training_label, spam_validation_features,
spam_validation_label = partition(spam_data, spam_labels, 0.2)

print(f"\nMNIST Training Features: {mnist_training_features.shape}")
print(f"MNIST Training Labels: {mnist_training_label.shape}")
print(f"MNIST Validation Features: {mnist_validation_features.shape}")
print(f"MNIST Validation Labels: {mnist_validation_label.shape}")
print(f"\nSpam Training Features: {spam_training_features.shape}")
print(f"Spam Training Labels: {spam_training_label.shape}")
print(f"Spam Validation Features: {spam_validation_features.shape}")
print(f"Spam Validation Labels: {spam_validation_label.shape}")


def accuracy_eval(true_labels, predicted_labels):
    if len(true_labels) != len(predicted_labels):
        raise ValueError("Inputs must have the same length!")
    total_labels = len(true_labels)

    # Calculate accuracy score
    correct_predictions = 0
    for i in range(total_labels):
        if true_labels[i] == predicted_labels[i]:
            correct_predictions += 1
    score = correct_predictions / total_labels

    return score
```

```
# Testing accuracy_eval:
true_labels = [1, 0, 2, 3, 3]
predicted_labels = [1, 0, 1, 0, 1]
score = accuracy_eval(true_labels, predicted_labels)
print(f"Accuracy: {score}")


#####################################################################
# QUESTION 4: support vector machines
from sklearn.svm import LinearSVC, SVC
import matplotlib.pyplot as plt

# training linear SVM on MNIST dataset
mnist_linear_svc = SVC(kernel='linear')
mnist_training_examples = [100, 200, 500, 1000, 2000, 5000, 10000]

# collect data on the accuracy of our model for the plot
mnist_training_accuracy = []
mnist_validation_accuracy = []

for limit in mnist_training_examples:
   # The below print statement is for debugging purposes:
   # print(f"Number of unique classes: {np.unique(example_training_label)}")

   # fit the SVM model to the given training data
   mnist_linear_svc.fit(mnist_training_features[:limit],
mnist_training_label.flatten()[:limit])

   training_score = accuracy_eval(mnist_training_label.flatten()[:limit],
mnist_linear_svc.predict(mnist_training_features[:limit]))
   validation_score = accuracy_eval(mnist_validation_label.flatten()[:limit],
mnist_linear_svc.predict(mnist_validation_features[:limit]))

   mnist_training_accuracy.append(training_score)
   mnist_validation_accuracy.append(validation_score)

   print(f"\nTraining with {limit} examples")
   print(f"Training accuracy: {training_score}")
   print(f"Validation Accuracy: {validation_score}")

# plotting accuracy
```

```
plt.plot(mnist_training_examples, mnist_training_accuracy, label="Training Accuracy",
marker='o')
plt.plot(mnist_training_examples, mnist_validation_accuracy, label="Validation
Accuracy", marker='o')
plt.xlabel("Number of Training Examples")
plt.ylabel("Accuracy")
plt.title("SVM Accuracy for MNIST dataset")
plt.legend()
plt.show()


# training linear SVM on Spam dataset
spam_linear_svc = SVC(kernel='linear')

ALL = 4171
spam_training_examples = [100, 200, 500, 1000, 2000, ALL]

spam_training_accuracy = []
spam_validation_accuracy = []

for limit in spam_training_examples:
   spam_linear_svc.fit(spam_training_features[:limit],
spam_training_label.flatten()[:limit])

   training_score = accuracy_eval(spam_training_label.flatten()[:limit],
spam_linear_svc.predict(spam_training_features[:limit]))
   validation_score = accuracy_eval(spam_validation_label.flatten()[:limit],
spam_linear_svc.predict(spam_validation_features[:limit]))

   spam_training_accuracy.append(training_score)
   spam_validation_accuracy.append(validation_score)

   print(f"\nTraining with {limit} examples")
   print(f"Training accuracy: {training_score}")
   print(f"Validation Accuracy: {validation_score}")

# plotting accuracy
plt.plot(spam_training_examples, spam_training_accuracy, label="Training Accuracy",
marker='o')
plt.plot(spam_training_examples, spam_validation_accuracy, label="Validation
Accuracy", marker='o')
plt.xlabel("Number of Training Examples")
```

```
plt.ylabel("Accuracy")
plt.title("SVM Accuracy for Spam Dataset")
plt.legend()
plt.show()



#####################################################################
# QUESTION 5: hyperparameter tuning

# Geometric sequence of C-values to try:
# The values 15 I'm testing are: 1e-10, 1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3,
0.01, 0.1, 1, 10, 100, 1000, 10000
C_values = [10**i for i in range(-10, 4)]

best_accuracy = 0.0
best_C = None
limit = 10000  # required to train with at least 10,000 training examples

for C_value in C_values:
   mnist_C_svc = SVC(kernel='linear', C=C_value)
   mnist_C_svc.fit(mnist_training_features[:limit],
mnist_training_label.flatten()[:limit])

   training_score = accuracy_eval(mnist_training_label.flatten()[:limit],
mnist_C_svc.predict(mnist_training_features[:limit]))
   validation_score = accuracy_eval(mnist_validation_label.flatten()[:limit],
mnist_C_svc.predict(mnist_validation_features[:limit]))

   print(f"\nTraining with {C_value} C-value")
   print(f"Training accuracy: {training_score}")
   print(f"Validation Accuracy: {validation_score}")

   # calculate the average accuracy
   accuracy = (training_score + validation_score) / 2

   # determine which C_value results in a model with the highest average accuracy
   if accuracy > best_accuracy:
       best_accuracy = accuracy
       best_C = C_value

print(f"\nBest C value: {best_C}")
print(f"Best Validation Accuracy: {best_accuracy}")
```

```
# After running the above hyperparameter tuning algorithm, I found that:
best_C = 0.0001


#####################################################################
# QUESTION 6: k-fold cross-validation
# from sklearn.model_selection import KFold

# Geometric sequence of C-values to try
# The values 14 I'm testing are: 1e-10, 1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3,
0.01, 0.1, 1, 10, 100, 500
Ck_values = [10**i for i in range(-10, 2)]
Ck_values.append(500)

best_accuracy = 0.0
best_Ck = None

# perform 5-fold cross-validation
n_splits = 5
kf_indices = np.array_split(np.arange(len(spam_training_features)), n_splits)

for Ck_value in Ck_values:
   avg_accuracy = 0.0

   for index in range(n_splits):
       validation_set_index = kf_indices[index]
       training_set_index = np.concatenate([kf_indices[j] for j in range(n_splits) if
j != index]) # everything that's not in the validation set

       # train our model on k-1 sets
       spam_C_svc = SVC(kernel='linear', C=Ck_value)
       spam_C_svc.fit(spam_training_features[training_set_index],
spam_training_label[training_set_index].flatten())

       # validate our model on the kth set
       avg_accuracy +=
accuracy_eval(spam_training_label[validation_set_index].flatten(),
spam_C_svc.predict(spam_training_features[validation_set_index]))

   # the cross-validation accuracy we report is the accuracy averaged over the k
iterations
```

```
    avg_accuracy /= 5

    print(f"\nC = {Ck_value}")
    print(f"Cross-Validation Accuracy = {avg_accuracy}")

    if avg_accuracy > best_accuracy:
        best_accuracy = avg_accuracy
        best_Ck = Ck_value

print(f"\nBest C value: {best_Ck}")
print(f"Best Cross-Validation Accuracy: {best_accuracy}")

# After running the above cross-validation algorithm, I found that:
best_Ck = 500


#####################################################################
# QUESTION 7: predictions for Kaggle
import pandas as pd

# A code snippet to help you save your results into a kaggle accepted csv
# Usage: results_to_csv(clf.predict(X_test))
def results_to_csv(y_test, file_name):
    y_test = y_test.astype(int)
    df = pd.DataFrame({'Category': y_test})
    df.index += 1 # Ensures that the index starts at 1
    df.to_csv(file_name, index_label='Id')

# for MNIST dataset
# retraining our SVM model
mnist_linear_svc = SVC(kernel='linear', C=best_C)
limit = 30000

mnist_linear_svc.fit(mnist_training_features[:limit],
mnist_training_label.flatten()[:limit])

training_score = accuracy_eval(mnist_training_label.flatten(),
mnist_linear_svc.predict(mnist_training_features))
validation_score = accuracy_eval(mnist_validation_label.flatten(),
mnist_linear_svc.predict(mnist_validation_features))
print(f"Final training accuracy: {training_score}")
print(f"Final validation Accuracy: {validation_score}")
```

```
# save predictions to csv file
mnist_data = np.load(f"../data/mnist-data.npz")
mnist_test_data = mnist_data["test_data"]
mnist_test_predictions =
mnist_linear_svc.predict(mnist_test_data.reshape((mnist_test_data.shape[0], -1)))

results_to_csv(mnist_test_predictions, "MNIST-submission.csv")
print("MNIST data saved to submission.csv")



# for Spam dataset
# retraining our SVM model
spam_linear_svc = LinearSVC(dual="auto", C=best_Ck)
limit = 2000

spam_linear_svc.fit(spam_training_features[:limit], spam_training_label[:limit])

training_score = accuracy_eval(spam_training_label.flatten(),
spam_linear_svc.predict(spam_training_features))
validation_score = accuracy_eval(spam_validation_label.flatten(),
spam_linear_svc.predict(spam_validation_features))
print(f"Final training accuracy: {training_score}")
print(f"Final validation Accuracy: {validation_score}")

# saving predictions to .csv file
spam_data = np.load(f"../data/spam-data.npz")
spam_test_data = spam_data["test_data"]
spam_test_predictions = spam_linear_svc.predict(spam_test_data)

results_to_csv(spam_test_predictions, "Spam-submission.csv")
print("Spam data saved to submission.csv")
```