CS 189 HW 1

Students Who Helped: Suhrid Saha

"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."

Sign: Shrihan Agarwal

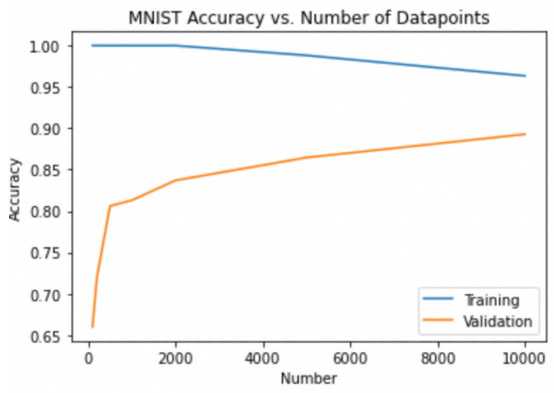
Q1.

```
def load_and_save(filename, num = None, percent = None):
    dataset = io.loadmat("data/" + filename)
    if percent:
        num = int(dataset["training_labels"].shape[0] * percent)
    t_dat, t_lbl, v_dat, v_lbl = split_data(dataset["training_data"],
                                            dataset["training_labels"],
                                            num)
    dataset["training_data"] = t_dat
    dataset["training_labels"] = t_lbl
    dataset["valid_data"] = v_dat
    dataset["valid_labels"] = v_lbl

    io.savemat("data/prep_" + filename, dataset)


def split_data(train, labels, num_valid):
    num_data = train.shape[0]
    assert num_valid <= len(train)
    assert num_data == labels.shape[0]
    idx = np.arange(num_data)
    np.random.shuffle(idx)
    train_shf = train[idx]
    lbl_shf = labels[idx]
    valid_dat = train_shf[:num_valid]
    valid_lbl = lbl_shf[:num_valid]
    train_dat = train_shf[num_valid:]
    train_lbl = lbl_shf[num_valid:]
    return train_dat, train_lbl, valid_dat, valid_lbl

load_and_save("mnist_data.mat", num = 10000)
load_and_save("spam_data.mat", percent = 0.2)
load_and_save("cifar10_data.mat", num = 5000)
```

This section of my code not only shuffles and splits the data, but also saves the data into a new .mat file for easy access. I use this later on in the code to streamline access to the modified data.

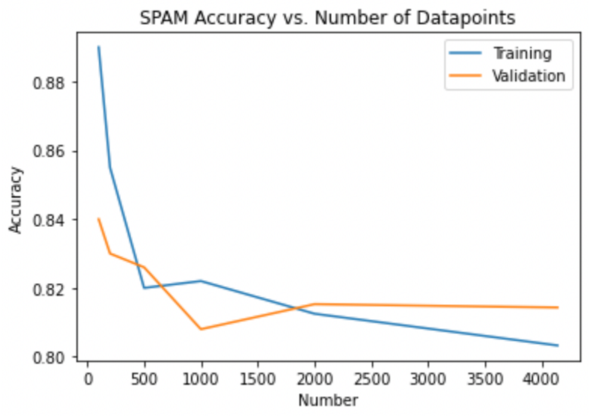
Q2.
(a)

As expected, validation accuracies are between 70-90%. The training accuracy drops with time as the linear svm has to accommodate more datapoints.



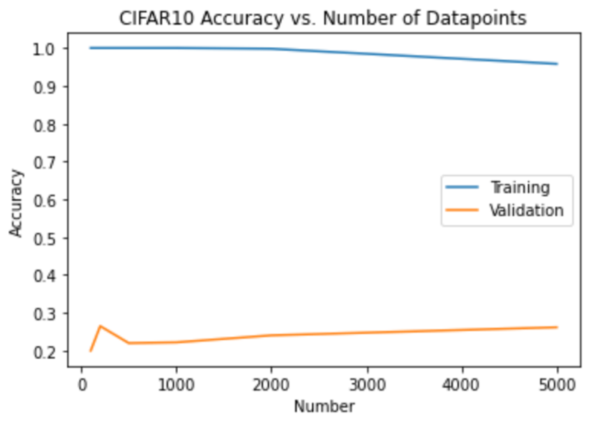
(b)

Similarly, the spam dataset maintains an accuracy between 70-90% as well. However, due to the stochasticity of the results for SPAM, k-fold cross validation is needed.



(c)

For CIFAR10, the dataset maintains an accuracy of around 25-35%. Code snippets for this question are in the APPENDIX.

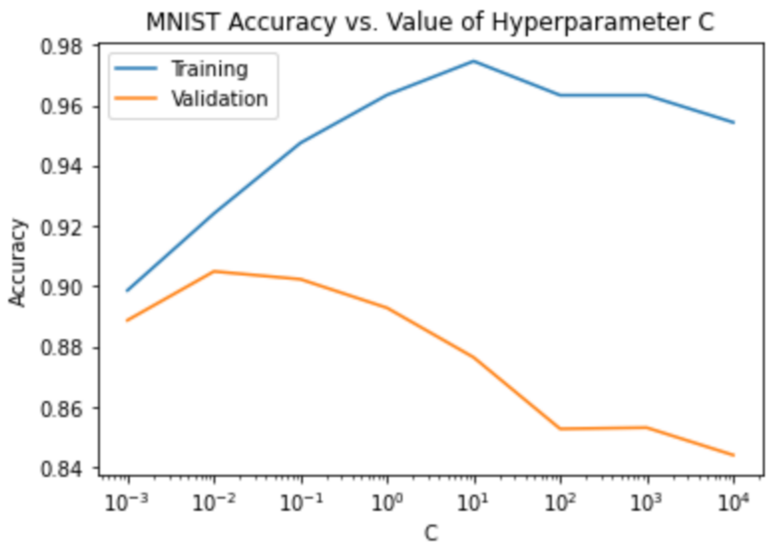


Q3.

The values tried are as follows. The best value is in bold, corresponding to a C of 10^{-2}.

Value of C	10^{-3}	**10^{-2}**	10^{-1}	10^{0}	10^{1}	10^{2}	10^{3}	10^{4}
Val Acc.	0.8887	**0.9049**	0.9023	0.8928	0.8764	0.8527	0.8531	0.8440
Train Acc.	0.8986	**0.9241**	0.9475	0.9634	0.9746	0.9633	0.9633	0.9543

These values were trained with a training set of 10,000 examples. And they are graphed below:

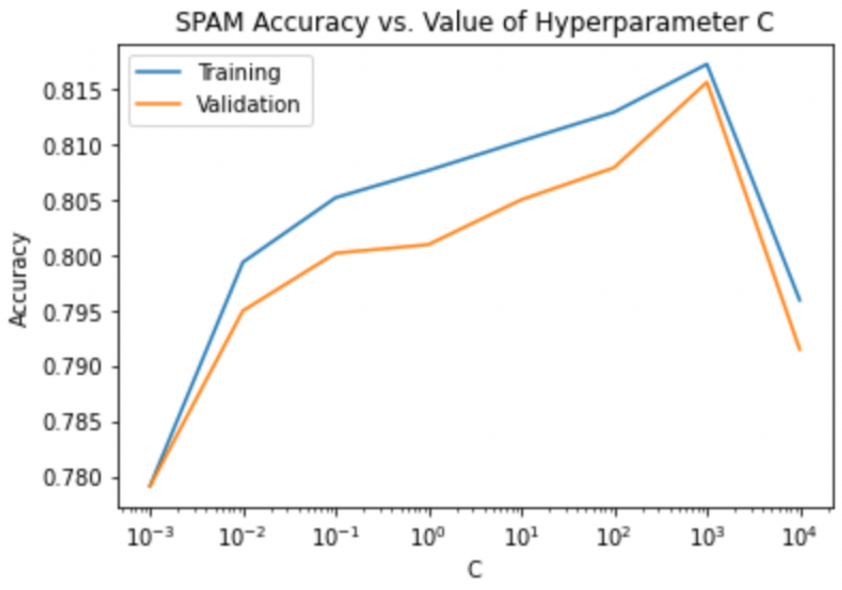


Q4.

Using 5-fold cross validation to get a more accurate accuracy for SPAM, the values tried are as follows. The best value is in bold, corresponding to a C of 10^3.

Value of C	10^{-3}	10^{-2}	10^{-1}	10^0	10^1	10^2	$\mathbf{10^3}$	10^4
Val Acc.	0.7791	0.7949	0.8002	0.8009	0.8050	0.8079	**0.8157**	0.7915
Train Acc.	0.7791	0.7994	0.8052	0.8076	0.8103	0.8130	**0.8173**	0.7960

The plot of the above data is below. Code for this is provided in the appendix.



Q5.

MNIST
Kaggle Score: 0.9833

- First, I tried standardizing the data by normalizing, i.e., dividing my 255. This created a minor difference, but not a significant one.
- Following the professor's suggestions on piazza, I attempted to standardize the standard deviation of the dataset in addition, but soon realized that that would not help, since the MNIST data doesn't follow a smooth normal distribution, rather, it has a few very bright points and other very dark points.
- Using this as a basis to go further, I attempted to increase the contrast. That is, I made all image points greater than 50 = 1, and all image points smaller than 50 = 0. This would ensure that only the shape of the number would be highlighted, and reduce complexity. Unfortunately, this change only reduced the performance of the algortihm.
- Finally, I switched to using sklearn's StandardSolver(), which standardizes the data for you. I did this on both the training and validation data. This boosted the algorithm's performance significantly.
- I then tried to increase the degree of the solver (polynomial rather than linear), and graphed the results for varying degrees. I also graphed a 2d optimization consisting of two hyperparameters: C and the degree. However, both cases showed that having a linear solver produced the best solution.
- I finally settled on 'rbf' rather than 'poly' for the solver type, and continued this for SPAM and CIFAR10 as well.
- I optimized C using a simple plot and choose technique. I finally settled on a C value of 1e5.
- I then trained on the entire MNIST dataset, including the validation data, and predicted values for the test data.

SPAM
Kaggle Score: 0.89038

- I added a few features manually, like "click", "money", etc., but quickly realized this was not efficient. I decided to find the optimal features using sklearn's TfidfVectorizer.
- I combed through the entire spam email dataset, searched for words, and added them to the vectorizer. I finally made it output the 100 best features for this dataset. I did the same for the ham dataset and took the intersection of the features by combining them and using a set.
- I copy-pasted this set to the featurize.py code, and coded it so that each of the features were added to the feature vector. The final length of the feature vector was 156.
- I then performed standardization with StandardScaler(), and optimized for the hyperparameter C.

- This got me to a local training accuracy of 0.9535 with no k-fold cross validation. I saved this and sent it to Kaggle, where it unfortunately performed worse due to the stochasticity.

CIFAR10
Kaggle Score: 0.55779

- I did not work too hard for this one. I simply standardized the set with StandardSolver(), optimized for C, and then most importantly, let my computer run for a few hours on the entire CIFAR10 dataset.
- Running on the entire dataset was sufficient to boost the performance from around 33% to 55%.

CS 189 HW1

Q6.

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

$$= \max_{\lambda_i \geq 0} \left[\text{result of minimization} \right]$$

Minimizing:

$$\frac{\partial}{\partial w}\left[|w|^2 - \sum_{i=1}^{n} \lambda_i (y_i (X_i \cdot w + \alpha) - 1) \right] = 0 \quad \textcircled{1}$$

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

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

This critical point is an optimum as the function is convex. The function is convex as the function is a norm of a linear function of vectors.

Let $A = \begin{bmatrix} y_1 X_{11} & y_2 X_{21} & \cdots \\ y_1 X_{12} & \ddots & \\ \vdots & & \\ y_1 X_1 & - & \end{bmatrix}$

$$\Rightarrow w = \frac{1}{2} A\lambda \qquad w^T = \frac{1}{2} \lambda^T A^T$$

Page 2 Q6. contd.

(a) contd.

$$|w|^2 = w^T w = \frac{1}{4} \lambda^T A^T A \lambda \quad \textcircled{3}$$

Also,

$$\frac{\partial}{\partial \alpha}\left[|w|^2 - \sum_{i=1}^{n} \lambda_i (y_i (x_i \cdot w + \alpha) - 1) \right] = 0 \quad \textcircled{2}$$

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

This is an optimum as it is the norm of a linear function (convex).

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

This constraint comes from minimizing α.

Plugging in $|w|^2$ from $\textcircled{3}$ into $\textcircled{0}$:

$$= \max_{\lambda_i \geq 0} \frac{1}{2} \lambda^T A^T A \lambda - \sum_{i=1}^{n} (\lambda_i y_i x_i \cdot w)$$

$$- \cancel{\sum \lambda_i y_i \alpha}^{0} + \text{[illegible]} + \sum_{i=1}^{n} \lambda_i \quad (\text{from } \textcircled{2})$$

$$= \max_{\lambda_i \geq 0} \frac{1}{4} \lambda^T A^T A \lambda - \sum_{i=1}^{n} \lambda_i y_i w^T x_i + \sum_{i=1}^{n} \lambda_i$$

$$= \max_{\lambda_i \geq 0} \frac{1}{4} \lambda^T A^T A \lambda - w^T \sum_{i=1}^{n} \lambda_i y_i x_i + \sum_{i=1}^{n} \lambda_i$$

Page 3 Q6. contd.

(a) contd.

$$= \max_{\lambda_i \geq 0} \; |w|^2 - 2w^T w + \sum_{i=1}^{n} \lambda_i$$

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

~~This point is an optima as it is the only local optimum.~~

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

We know $w = \frac{1}{2} A\lambda = \frac{1}{2} \sum \lambda_i y_i X_i$ (from a)

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

$$\boxed{= \begin{cases} +1 & \text{if } \alpha^* + \frac{1}{2} \sum \lambda_i^* y_i X_i \cdot x \\ -1 & \text{otherwise} \end{cases}}$$

(c) If $\lambda_i^* > 0$, this implies

$$y_i (x_i \cdot w^* + \alpha^*) - 1 = 0$$

$$\Rightarrow X_i \circ w^* + \alpha^* = \frac{1}{y_i} = y_i \quad \left(\boxed{y_i = \pm 1}\right)$$

PTO

~~This is identical to points on the margin~~

Page 4 Q6 contd.

(c) contd.

~~Repeated~~:
This is identical to points on the margin, which satisfy $x \cdot w + \alpha = \pm 1$.
Therefore, we can say that for $\lambda_i > 0$ the points X_i are ~~[illegible]~~ on the margins.

(d) The decision rule is:

$$\alpha^* + \frac{1}{2} \sum_{i=1}^{n} \lambda_i^* y_i X_i \cdot x \geq 0 \quad ①$$

For non-support vectors, $\lambda_i \leq 0$.
But, λ_i must be ≥ 0 by the constraints on the dual. Therefore, non-support vectors have $\lambda_i = 0$. Therefore, their decision rule is trivial:

$\lambda_i = 0$:

$$① \rightarrow \alpha^* + \frac{1}{2} \cancel{\sum_{i=1}^{n} \lambda_i^* y_i X_i \cdot x}^{\,0} \geq 0$$

$$\Rightarrow \alpha^* \geq 0$$

This is either true for all non-support vectors, or false, depending on α^*.
Therefore, we only need the support

Page 5 Q6. contd.

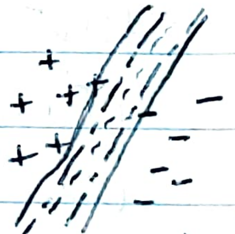
(d) vectors to calculate the decision
(contd.) rule, since for them it is not trivial.

(e) $w' = \frac{w}{1+\epsilon/2}$, new bias $= \alpha'$

$\epsilon > 0$.

Structure of Argument:

Assume there are no support vectors.
Then, the margin can be expanded until it ~~touches~~ has a support vector, and still be valid, producing a more optimal solution (larger margin).



Therefore, the optimal margin must have a support vector.

Math: (For the positive class)
Let the original margin be $X_i \cdot w + \alpha = \underset{\lambda_i}{\pm 1}$ but without any support vectors. We can construct a new margin with $w' = w/_{1+\epsilon/2}$, and a new α'.

$w' = w/(1+\epsilon/2)$. Let $\epsilon = X_i \cdot w + \alpha - 1$ (distance to margin)
w' is then reduced by ϵ, and the with of the margin is inversely proportional to w.
By choosing ϵ like this, we increase the width

by the distance to the nearest point of the +1 class.

$$\varepsilon = X_i \cdot \omega + \alpha - 1$$

$$\omega' = \frac{\omega}{1 + \epsilon/2} = \frac{\omega}{\left(1 + \left(\frac{X_i \cdot \omega + \alpha - 1}{2}\right)\right)}$$

Lastly, we want to bias α' such that the closest point is a support vector:

Let X_1 be the ~~support~~ closest point.

We want:

$$X_1 \cdot \omega' + \alpha' = 1$$

$$\Rightarrow \alpha' = 1 - X_1 \cdot \omega'$$

$$= \boxed{1 - X_1 \cdot \left(\frac{\omega}{1 + \left(\frac{X_i \cdot \omega + \alpha - 1}{2}\right)}\right)}$$

By choosing α' as the above, and ω' as $\omega/(1+\epsilon/2)$, we have widened the margin. Therefore, we have found a more optimal margin and value for ω. This contradicts our assumption of having an optimal margin at the start. Since this argument is symmetric, it is true for both classes.

Appendix

Q1.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.metrics import accuracy_score
from scipy import io


# Q1
def load_and_save(filename, num = None, percent = None):
    dataset = io.loadmat("data/" + filename)
    if percent:
        num = int(dataset["training_labels"].shape[0] * percent)
    t_dat, t_lbl, v_dat, v_lbl = split_data(dataset["training_data"],
                                            dataset["training_labels"],
                                            num)
    dataset["training_data"] = t_dat
    dataset["training_labels"] = t_lbl
    dataset["valid_data"] = v_dat
    dataset["vaild_labels"] = v_lbl

    io.savemat("data/prep_" + filename, dataset)


def split_data(train, labels, num_valid):
    num_data = train.shape[0]
    assert num_valid <= len(train)
    assert num_data == labels.shape[0]
    idx = np.arange(num_data)
    np.random.shuffle(idx)
    train_shf = train[idx]
    lbl_shf = labels[idx]
    valid_dat = train_shf[:num_valid]
    valid_lbl = lbl_shf[:num_valid]
    train_dat = train_shf[num_valid:]
    train_lbl = lbl_shf[num_valid:]
    return train_dat, train_lbl, valid_dat, valid_lbl

load_and_save("mnist_data.mat", num = 10000)
load_and_save("spam_data.mat", percent = 0.2)
load_and_save("cifar10_data.mat", num = 5000)
```

Q2.

```
# Q2

def perform_training_mnist(data, num):
    td = data["training_data"][:num] / 255
    tl = np.ravel(data["training_labels"])[:num]
    vd = data["valid_data"][:num] / 255
    vl = np.ravel(data["valid_labels"])[:num]
    alg = svm.LinearSVC(max_iter = 5000)
    alg.fit(td, tl)
    vp = alg.predict(vd)
    tp = alg.predict(td)
    vacc = accuracy_score(vl, vp)
    tacc = accuracy_score(tl, tp)
    return num, vacc, tacc

def perform_training_spam(data, num):
    td = data["training_data"][:num]
    tl = np.ravel(data["training_labels"])[:num]
    vd = data["valid_data"][:num]
    vl = np.ravel(data["valid_labels"])[:num]
    alg = svm.LinearSVC(max_iter = 100000)
    alg.fit(td, tl)
    vp = alg.predict(vd)
    tp = alg.predict(td)
    vacc = accuracy_score(vl, vp)
    tacc = accuracy_score(tl, tp)
    return num, vacc, tacc

# MNIST
data = io.loadmat("data/prep_mnist_data.mat")
qt = [100, 200, 500, 1000, 2000, 5000, 10000]
tot_acc = np.array([perform_training_mnist(data, i) for i in qt])
```

```python
plt.plot(qt, tot_acc.T[2], label = "Training")
plt.plot(qt, tot_acc.T[1], label = "Validation")
plt.title("MNIST Accuracy vs. Number of Datapoints")
plt.legend()
plt.xlabel("Number")
plt.ylabel("Accuracy")

# SPAM
sata = io.loadmat("data/prep_spam_data.mat")
qt = [100, 200, 500, 1000, 2000, 4139]
tot_vacc = np.array([perform_training_spam(sata, i) for i in qt])
plt.plot(qt, tot_vacc.T[2], label = "Training")
plt.plot(qt, tot_vacc.T[1], label = "Validation")
plt.title("SPAM Accuracy vs. Number of Datapoints")
plt.legend()
plt.xlabel("Number")
plt.ylabel("Accuracy")

# CIFAR
cata = io.loadmat("data/prep_cifar10_data.mat")
qt = [100, 200, 500, 1000, 2000, 5000]
tot_acc = np.array([perform_training_mnist(cata, i) for i in qt])
plt.plot(qt, tot_acc.T[2], label = "Training")
plt.plot(qt, tot_acc.T[1], label = "Validation")
plt.title("CIFAR10 Accuracy vs. Number of Datapoints")
plt.legend()
plt.xlabel("Number")
plt.ylabel("Accuracy")
```

Q3.

```python
# Q3

def perform_training_mnist_C(data, num, C_val):
    td = data["training_data"][:num] / 255
    tl = np.ravel(data["training_labels"])[:num]
    vd = data["valid_data"][:num] / 255
    vl = np.ravel(data["valid_labels"])[:num]
    alg = svm.LinearSVC(max_iter = 5000, C = C_val)
    alg.fit(td, tl)
    vp = alg.predict(vd)
    tp = alg.predict(td)
    vacc = accuracy_score(vl, vp)
    tacc = accuracy_score(tl, tp)
    return num, vacc, tacc

C_vals = np.logspace(-3, 4, 8)

data = io.loadmat("data/prep_mnist_data.mat")
tot_acc = np.array([perform_training_mnist_C(data, 10000, i) for i in C_vals])
plt.plot(C_vals, tot_acc.T[2], label = "Training")
plt.plot(C_vals, tot_acc.T[1], label = "Validation")
plt.title("MNIST Accuracy vs. Value of Hyperparameter C")
plt.legend()
plt.xlabel("C")
plt.ylabel("Accuracy")
print("Validation Accuracy:", tot_acc.T[1])
print("Training Accuracy:", tot_acc.T[2])
```

Q4.

```python
# Q4

def split_data_kcross(train, labels, num_valid, n):
    num_data = train.shape[0]
    assert num_valid <= len(train)
    assert num_data == labels.shape[0]
    idx = np.arange(num_data)
    np.random.shuffle(idx)
    train_shf = train[idx]
    lbl_shf = labels[idx]
    valid_dat = train_shf[n * num_valid : (n + 1) * num_valid]
    valid_lbl = lbl_shf[n * num_valid : (n + 1) * num_valid]
    train_dat = np.concatenate([train_shf[:n * num_valid], train_shf[(n + 1) * num_valid:]])
    train_lbl = np.concatenate([lbl_shf[:n * num_valid], lbl_shf[(n + 1) * num_valid:]])
    return train_dat, train_lbl, valid_dat, valid_lbl

def perform_training_spam_cross(data, C_val):
    td = data[0]
    tl = np.ravel(data[1])
    vd = data[2]
    vl = np.ravel(data[3])
    alg = svm.LinearSVC(max_iter = 10000, C = C_val)
    alg.fit(td, tl)
    vp = alg.predict(vd)
    tp = alg.predict(td)
    vacc = accuracy_score(vl, vp)
    tacc = accuracy_score(tl, tp)
    return vacc, tacc

def averageTraining(total_data, C_val):
    res = np.array([perform_training_spam_cross(total_data[i], C_val) for i in range(5)])
    vacc = np.mean(res.T[0])
    tacc = np.mean(res.T[1])
    return vacc, tacc

spam_dat = io.loadmat("data/spam_data.mat")
total_data = [split_data_kcross(spam_dat["training_data"],
                                spam_dat["training_labels"],
                                spam_dat["training_data"].shape[0] // 5,
                                i) for i in range(5)]

C_vals = np.logspace(-3, 4, 8)
res = np.array([averageTraining(total_data, i) for i in C_vals])
plt.plot(C_vals, res.T[1], label = "Training")
plt.plot(C_vals, res.T[0], label = "Validation")
plt.title("SPAM Accuracy vs. Value of Hyperparameter C")
plt.legend()
plt.semilogx()
plt.xlabel("C")
plt.ylabel("Accuracy")
```

Q5.

```
# Q5

import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.metrics import accuracy_score
from scipy import io

def split_data(train, labels, num_valid):
    num_data = train.shape[0]
    assert num_valid <= len(train)
    assert num_data == labels.shape[0]
    idx = np.arange(num_data)
    np.random.shuffle(idx)
    train_shf = train[idx]
    lbl_shf = labels[idx]
    valid_dat = train_shf[:num_valid]
    valid_lbl = lbl_shf[:num_valid]
    train_dat = train_shf[num_valid:]
    train_lbl = lbl_shf[num_valid:]
    return train_dat, train_lbl, valid_dat, valid_lbl

def load_and_save(filename, num = None, percent = None):
    dataset = io.loadmat(filename)
    if percent:
        num = int(dataset["training_labels"].shape[0] * percent)
    t_dat, t_lbl, v_dat, v_lbl = split_data(dataset["training_data"],
                                            dataset["training_labels"],
                                            num)

    dataset["training_data"] = t_dat
    dataset["training_labels"] = t_lbl
    dataset["valid_data"] = v_dat
    dataset["valid_labels"] = v_lbl

    io.savemat("prep_" + filename, dataset)

load_and_save("mnist_kdata.mat", num = 10000)

def perform_training_mnist(data, num):
    td = data["training_data"][:num] / 255
    tl = np.ravel(data["training_labels"])[:num]
    vd = data["valid_data"][:num] / 255
    vl = np.ravel(data["valid_labels"])[:num]
    alg = svm.LinearSVC(max_iter = 5000)
    alg.fit(td, tl)
    vp = alg.predict(vd)
    tp = alg.predict(td)
    vacc = accuracy_score(vl, vp)
    tacc = accuracy_score(tl, tp)
    return num, vacc, tacc
```

```python
from sklearn.preprocessing import StandardScaler

def perform_training_mnist_2(data, num, C_val):

    td = data["training_data"][:num]
    tl = np.ravel(data["training_labels"])[:num]
    vd = data["valid_data"][:num]
    vl = np.ravel(data["valid_labels"])[:num]

    scaler = StandardScaler()
    scaler.fit_transform(td)
    scaler.transform(vd)

    alg = svm.SVC(max_iter = 5000, kernel = 'rbf', degree = 1, C = 1e5)
    alg.fit(td, tl)
    vp = alg.predict(vd)
    tp = alg.predict(td)
    vacc = accuracy_score(vl, vp)
    tacc = accuracy_score(tl, tp)
    return vacc


def perform_training_mnist_3(data):

    td = data["training_data"]
    tl = np.ravel(data["training_labels"])
    vd = data["valid_data"]
    vl = np.ravel(data["valid_labels"])

    scaler = StandardScaler()
    scaler.fit_transform(td)
    scaler.transform(vd)

    alg = svm.SVC(max_iter = 5000, kernel = 'rbf', degree = 1, C = 1e5)
    alg.fit(td, tl)
    vp = alg.predict(vd)
    tp = alg.predict(td)
    vacc = accuracy_score(vl, vp)
    tacc = accuracy_score(tl, tp)
    return vacc

def perform_training_mnist_final(data):

    td = data["training_data"]
    tl = np.ravel(data["training_labels"])
    vd = data["test_data"]

    scaler = StandardScaler()
    scaler.fit_transform(td)
    scaler.transform(vd)

    alg = svm.SVC(max_iter = 5000, kernel = 'rbf', degree = 1, C = 1e5)
    alg.fit(td, tl)
    vp = alg.predict(vd)
    return alg, vp

#data = io.loadmat("prep_mnist_kdata.mat")
#C_vals = np.logspace(-5, 5, 10)
#C_vals = np.linspace(2, 2.3, 30)
#tot_acc = np.array([perform_training_mnist_2(data, 1000, i) for i in C_vals])
# 2D optimization of polynomial degree and C (training was tried with kernel = 'poly')
# deg = np.linspace(1, 10, 10)
# tot_acc = np.array([[perform_training_mnist_2(data, 10000, i, j) for i in C_vals] for j in deg])

#plt.plot(C_vals, tot_acc.T[2] - 0.86, label = "Training")
plt.plot(C_vals, tot_acc.T, label = "Validation")
plt.title("MNIST Accuracy vs. Value of Degree")
plt.legend()
plt.semilogx()
plt.xlabel("C")
plt.ylabel("Accuracy")

data = io.loadmat("mnist_kdata.mat")
alg, vp = perform_training_mnist_final(data)
idx = list(range(1, len(vp) + 1))
pre_csv = np.array(list(zip(idx, vp)), dtype = int)
np.savetxt("shri_pred.csv", pre_csv, fmt = '%s', delimiter=',')

from collections import defaultdict
import glob
import re
import scipy.io
import numpy as np
spam_filenames = glob.glob('data/spam/' + '*.txt')
ham_filenames = glob.glob('data/ham/' + '*.txt')
```

```python
email_list = []
for filename in spam_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

            # Create a feature vector
            email_list.append(text)

from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(max_features = 100)
X = vectorizer.fit_transform(email_list)
a = vectorizer.get_feature_names_out()


email_list_ham = []
for filename in ham_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

            # Create a feature vector
            email_list_ham.append(text)


vectorizer = TfidfVectorizer(max_features = 100)
X = vectorizer.fit_transform(email_list_ham)
b = vectorizer.get_feature_names_out()
c = np.concatenate([a, b])
final_feature_list = set(c)

# Feature set was modified as follows
for i in feature_set:
    a = lambda text, freq: float(freq[i])
    feature.append(a)

sata = io.loadmat('data/spam_data.mat')

def split_data(train, labels, num_valid):
    num_data = train.shape[0]
    assert num_valid <= len(train)
    assert num_data == labels.shape[0]
    idx = np.arange(num_data)
    np.random.shuffle(idx)
    train_shf = train[idx]
    lbl_shf = labels[idx]
    valid_dat = train_shf[:num_valid]
    valid_lbl = lbl_shf[:num_valid]
    train_dat = train_shf[num_valid:]
    train_lbl = lbl_shf[num_valid:]
    return train_dat, train_lbl, valid_dat, valid_lbl

def load_and_save(filename, num = None, percent = None):
    dataset = io.loadmat(filename)
    if percent:
        num = int(dataset["training_labels"].shape[0] * percent)
    t_dat, t_lbl, v_dat, v_lbl = split_data(dataset["training_data"],
                                            dataset["training_labels"],
                                            num)
    dataset["training_data"] = t_dat
    dataset["training_labels"] = t_lbl
    dataset["valid_data"] = v_dat
    dataset["valid_labels"] = v_lbl

    io.savemat("prep_"+filename, dataset)

def perform_training_spam(data, num, C_val):

    td = data["training_data"][:num]
    tl = np.ravel(data["training_labels"])[:num]
    vd = data["valid_data"][:num]
    vl = np.ravel(data["valid_labels"])[:num]

    scaler = StandardScaler()
    scaler.fit_transform(td)
    scaler.transform(vd)

    alg = svm.SVC(max_iter = 10000, kernel = 'rbf', C = C_val)
    alg.fit(td, tl)
    vp = alg.predict(vd)
    tp = alg.predict(td)
    vacc = accuracy_score(vl, vp)
    tacc = accuracy_score(tl, tp)
    return vacc

def perform_training_spam_final(data):

    td = data["training_data"]
    tl = np.ravel(data["training_labels"])
    test = data["test_data"]

    scaler = StandardScaler()
    scaler.fit_transform(td)
    scaler.transform(test)

    alg = svm.SVC(max_iter = 10000, kernel = 'rbf', C = 14.3)
    alg.fit(td, tl)
    vp = alg.predict(test)
    return alg, vp

load_and_save("spam_data.mat", percent = 0.2)
sata_fin = io.loadmat("prep_spam_data.mat")

from sklearn.preprocessing import StandardScaler
#C_vals = np.logspace(-5, 10, 10)
#C_vals = np.logspace(0.6, 1.4, 10)
C_vals = np.linspace(10, 16, 10)
res = [perform_training_spam(sata_fin, 5000, i) for i in C_vals]

res = np.array(res)
plt.plot(C_vals, res)
plt.semilogx()

data_final = io.loadmat("spam_data.mat")
alg, pred = perform_training_spam_final(data_final)
idx = list(range(1, len(pred) + 1))
pre_csv = np.array(list(zip(idx, pred)), dtype = int)
np.savetxt("shri_pred_spam.csv", pre_csv, fmt = '%s', delimiter=',')


def split_data(train, labels, num_valid):
    num_data = train.shape[0]
    assert num_valid <= len(train)
    assert num_data == labels.shape[0]
    idx = np.arange(num_data)
    np.random.shuffle(idx)
    train_shf = train[idx]
    lbl_shf = labels[idx]
    valid_dat = train_shf[:num_valid]
    valid_lbl = lbl_shf[:num_valid]
    train_dat = train_shf[num_valid:]
    train_lbl = lbl_shf[num_valid:]
    return train_dat, train_lbl, valid_dat, valid_lbl

def load_and_save(filename, num = None, percent = None):
    dataset = io.loadmat(filename)
    if percent:
        num = int(dataset["training_labels"].shape[0] * percent)
    t_dat, t_lbl, v_dat, v_lbl = split_data(dataset["training_data"],
                                            dataset["training_labels"],
                                            num)
    dataset["training_data"] = t_dat
    dataset["training_labels"] = t_lbl
    dataset["valid_data"] = v_dat
    dataset["valid_labels"] = v_lbl

    io.savemat("prep_"+filename, dataset)

def perform_training_cifar(data, num, C_val):

    td = data["training_data"][:num]
    tl = np.ravel(data["training_labels"])[:num]
    vd = data["valid_data"][:num]
    vl = np.ravel(data["valid_labels"])[:num]

    scaler = StandardScaler()
    scaler.fit_transform(td)
    scaler.transform(vd)

    alg = svm.SVC(max_iter = 5000, kernel = 'rbf', C = C_val)
    alg.fit(td, tl)
    vp = alg.predict(vd)
    tp = alg.predict(td)
    vacc = accuracy_score(vl, vp)
    tacc = accuracy_score(tl, tp)
    return vacc

def perform_training_cifar_final(data, C_val):

    td = data["training_data"]
    tl = np.ravel(data["training_labels"])
    vd = data["test_data"]

    scaler = StandardScaler()
    scaler.fit_transform(td)
    scaler.transform(vd)

    alg = svm.SVC(max_iter = 5000, kernel = 'rbf', C = C_val)
    alg.fit(td, tl)
    vp = alg.predict(vd)
    return alg, vp

load_and_save("cifar10_data.mat", num = 5000)
cata = io.loadmat("prep_cifar10_data.mat")
#C_vals = np.logspace(-5, 5, 10)
C_vals = np.logspace(0, 30, 10)
res = np.array([perform_training_cifar(cata, 1000, i) for i in C_vals])
plt.plot(C_vals, res)
plt.semilogx()
cata_fin = io.loadmat("cifar10_data.mat")
alg, vp = perform_training_cifar_final(cata_fin, 1e7)
idx = list(range(1, len(vp) + 1))
pre_csv = np.array(list(zip(idx, vp)), dtype = int)
np.savetxt("shri_pred_cifar.csv", pre_csv, fmt = '%s', delimiter=',')
```