Project 1 - Support Vector Machine Classification

NAME(S): Matthew Wong, Bill Xia

DATE:

What will we do?

Using gradient descent, we will build a Support Vector Machine to find the optimal hyperplane that maximizes the margin between two toy data classes.

What are some use cases for SVMs?

-Classification, regression (time series prediction, etc.), outlier detection, clustering

How does an SVM compare to other ML algorithms?

alt text Classifiers: (a) Logistic Regression, (b) SVM, and (c) Multi-Layer Perception (MLP)

- As a rule of thumb, SVMs are great for relatively small data sets with fewer outliers.
- Other algorithms (Random forests, deep neural networks, etc.) require more data but almost always develop robust models.
- The decision of which classifier to use depends on your dataset and the general complexity of the problem.
- "Premature optimization is the root of all evil (or at least most of it) in programming." - Donald Knuth, CS Professor (Turing award speech 1974)

Other Examples

- Learning to use Scikit-learn's SVM function to classify images https://github.com/ksopyla/svm_mnist_digit_classification
- Pulse classification, more useful dataset https://github.com/akasantony/pulse-classification-svm

What is a Support Vector Machine?

It's a supervised machine learning algorithm that can be used for both classification and regression problems. But it's usually used for classification. Given two or more labeled data classes, it acts as a discriminative classifier, formally defined by an optimal hyperplane that separates all the classes. New examples mapped into that space can then be categorized based on which side of the gap they fall.

What are Support Vectors?

alt text

Support vectors are the data points nearest to the hyperplane, the points of a data set that, if removed, would alter the position of the dividing hyperplane. Because of this, they can be considered the critical elements of a data set; they help us build our SVM.

What is a hyperplane?

alt text

Geometry tells us that a hyperplane is a subspace of one dimension less than its ambient space. For instance, a hyperplane of an n-dimensional space is a flat subset with size $n-1$. By its nature, it separates the space in half.

Linear vs nonlinear classification?

Sometimes our data is linearly separable. That means for N classes with M features. We can learn a mapping that is a linear combination. (like $y=mx+b$). Or even a multidimensional hyperplane ($y=x+z+b+q$). No matter how many dimensions/features a set of classes have, we can represent the mapping using a linear function.

But sometimes it is not. Like if there was a quadratic mapping. Luckily for us, SVMs can efficiently perform a non-linear classification using what is called the kernel trick.

alt text

More on this as a Bonus question comes at the end of notebook.

All right, let's get to the building!

Instructions

In this assignment, you will implement a support vector machine (SVM) from scratch, and you will use your implementation for multiclass classification on the MNIST dataset.

In `implementation.py` implement the SVM class. In the fit function, use `scipy.minimize` (see documentation) to solve the constrained optimization problem:

$$\underset{a}{\text{maximize}} \quad \sum_{i=1}^{n} a_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j y_i y_j (x_i \cdot x_j)$$
$$\text{subject to} \quad a_i \geq 0, i = 1, \ldots, n$$

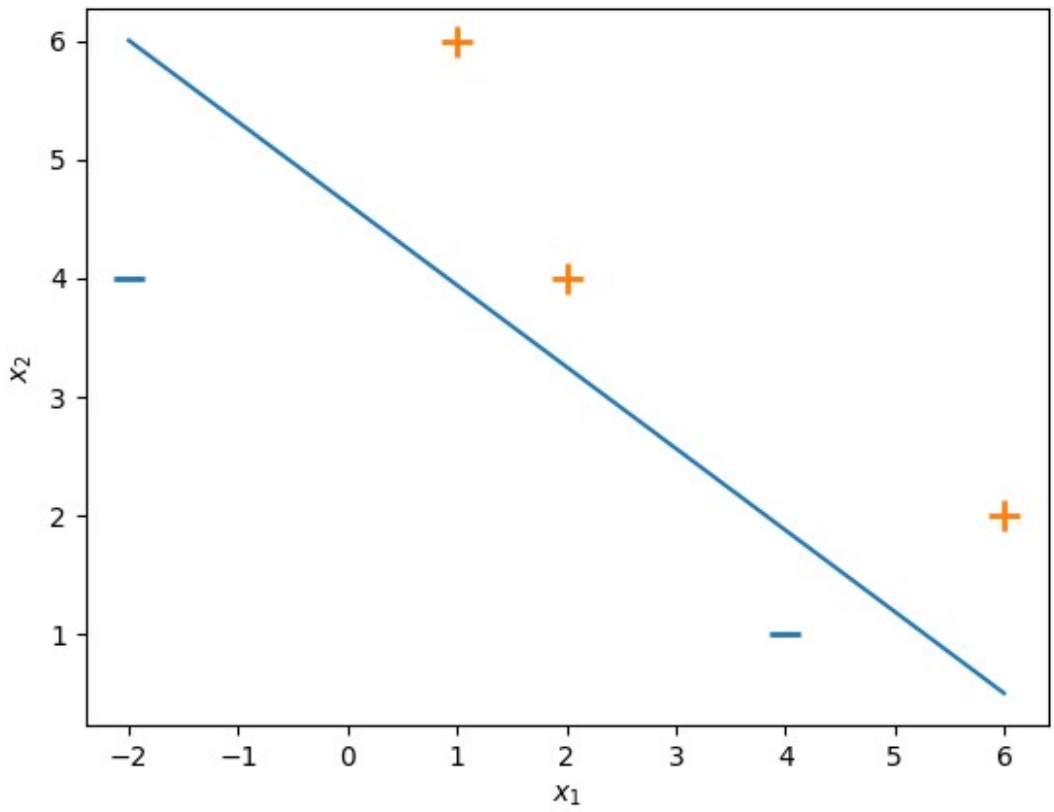
Note: An SVM is a convex optimization problem. Using to solve the equation above will be computationally expensive given larger datasets. CS 168 Convex Optimization is a course to take later if interested in optimization and the mathematics and intuition that drives it.

```
import numpy as np
import pandas as pd

from scipy.io import loadmat
```

```
from implementation import SVM, linear_kernel, nonlinear_kernel
# from solution import SVM, linear_kernel, nonlinear_kernel
from sklearn.datasets import make_blobs
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix

import matplotlib
import matplotlib.pyplot as plt

%load_ext autoreload
%autoreload 2
```

Step 1 - Get Data

```
# Input data - of the form [Bias term, x_1 value, x_2 value]
X = np.array([
    [1, -2, 4,],
    [1, 4, 1,],
    [1, 1, 6,],
    [1, 2, 4,],
    [1, 6, 2,],
])

# Associated output labels - first 2 examples are labeled '-1' and
last 3 are labeled '+1'
y = np.array([-1,-1,1,1,1])

# Let's plot these examples on a 2D graph!
# Plot the negative samples (the first 2)
plt.scatter(X[:,1][y==-1], X[:,2][y==-1], s=120, marker='_',
linewidths=2)
# Plot the positive samples (the last 3)
plt.scatter(X[:,1][y==1], X[:,2][y==1], s=120, marker='+',
linewidths=2)

# Print a possible hyperplane, that is separating the two classes.
# we'll two points and draw the line between them (naive guess)
plt.plot([-2,6],[6,0.5])
plt.xlabel(r"$x_1$")
plt.ylabel(r"$x_2$")
plt.show()
```



SVM basics

SVM using scikit-learn.

```
result = SVC(kernel="linear")
result.fit(X, y.ravel())

print("scikit-learn indices of support vectors:", result.support_)
```

```
scikit-learn indices of support vectors: [0 1 3 4]
```

Implement and test SVM to sklearn's version (20 points)

Compare the indices of support vectors from scikit-lean with `implementation.py` using toy data.

```
# TODO: implement SVM, along with linear_kernel

result = SVC(kernel="linear")
result.fit(X, y)

print("scikit-learn indices of support vectors:", result.support_)

svm = SVM(kernel=linear_kernel)
svm.fit(X, y)
```

```
print()
print("Alpha", svm.a)
print("Weights", svm.w)
print("Bias", svm.b)

print()
print("SVC Weights", result.coef_)
```

```
scikit-learn indices of support vectors: [0 1 3 4]
Optimization terminated successfully

Alpha [ 1.47549948e-01  4.77448333e-01 -3.96384314e-15  4.08827516e-01
  2.16170765e-01]
Weights [4.02455846e-15 4.99986188e-01 1.00000347e+00]
Bias -3.9999827826965246

SVC Weights [[0.         0.5        0.99969451]]
```

```
print("implementation.py indices of support vectors:", \
      np.array(range(y.shape[0]))[svm.a>1e-8])
if (result.support_ != np.array(range(y.shape[0]))[svm.a>1e-8]).all():
    raise Exception("The calculation is wrong")
```

```
implementation.py indices of support vectors: [0 1 3 4]
```

Compare the weights assigned to the features from scikit-lean with `implementation.py`.

```
# TODO - other sections were done for you, specify the variables to
print,
# find the difference, and check it is within reasonable error from
that of
# sklearn's version.

# print("scikit-learn weights assigned to the features:", VAR)
# print("implementation.py weights assigned to the features:", VAR)


print("scikit-learn weights assigned to the features:", result.coef_)
print("implementation.py weights assigned to the features:", svm.w)

diff = abs(result.coef_ - svm.w)
if (diff > 1e-3).any():
    raise Exception("The calculation is wrong")
```

```
scikit-learn weights assigned to the features: [[0.         0.5
0.99969451]]
implementation.py weights assigned to the features: [4.02455846e-15
4.99986188e-01 1.00000347e+00]
```

Compare the bias weight from scikit-lean with `implementation.py`.

```
print("scikit-learn bias weight:", result.intercept_)
print("implementation.py bias weight:", svm.b)

diff = abs(result.intercept_ - svm.b)
if (diff > 1e-3).all():
    raise Exception("The calculation is wrong")
```

```
scikit-learn bias weight: [-3.99915989]
implementation.py bias weight: -3.9999827826965246
```

Compare the predictions from scikit-lean with `implementation.py`.

```
X_test = np.array([
    [4, 4, -1],
    [1, 3, -1]
    ])
print("scikit-learn predictions:", result.predict(X_test))
print("implementation.py predictions:", svm.predict(X_test))

if (svm.predict(X_test) != result.predict(X_test)).all():
    raise Exception("The calculation is wrong")
```

```
scikit-learn predictions: [-1 -1]
implementation.py predictions: [-1 -1]
```

Using SKLearns SVM (*one-versus-the-rest*)

You can load the data with `scipy.io.loadmat`, which will return a Python dictionary containing the test and train data and labels.

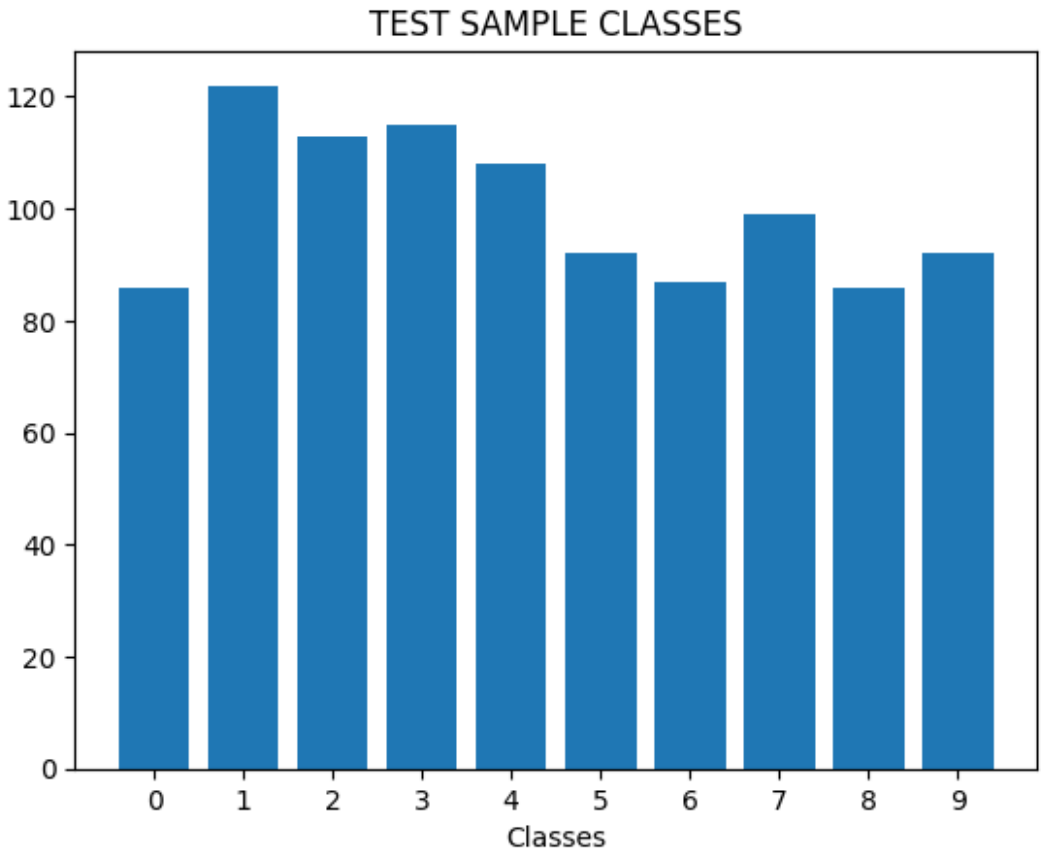
```
mnist = loadmat('data/MNIST.mat')
train_samples = mnist['train_samples']
train_samples_labels = mnist['train_samples_labels']
test_samples = mnist['test_samples']
test_samples_labels = mnist['test_samples_labels']
```

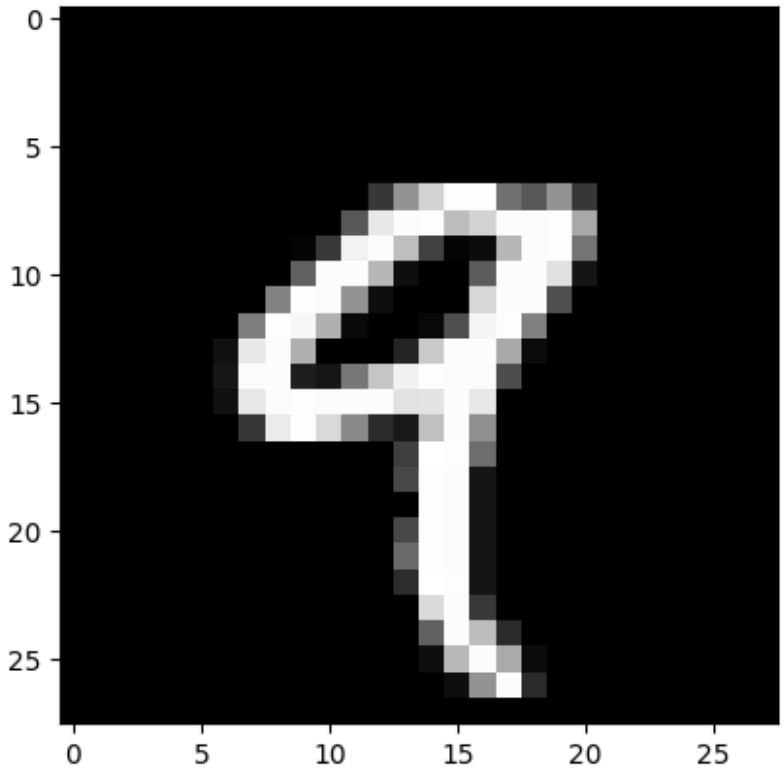
Explore the MNIST dataset

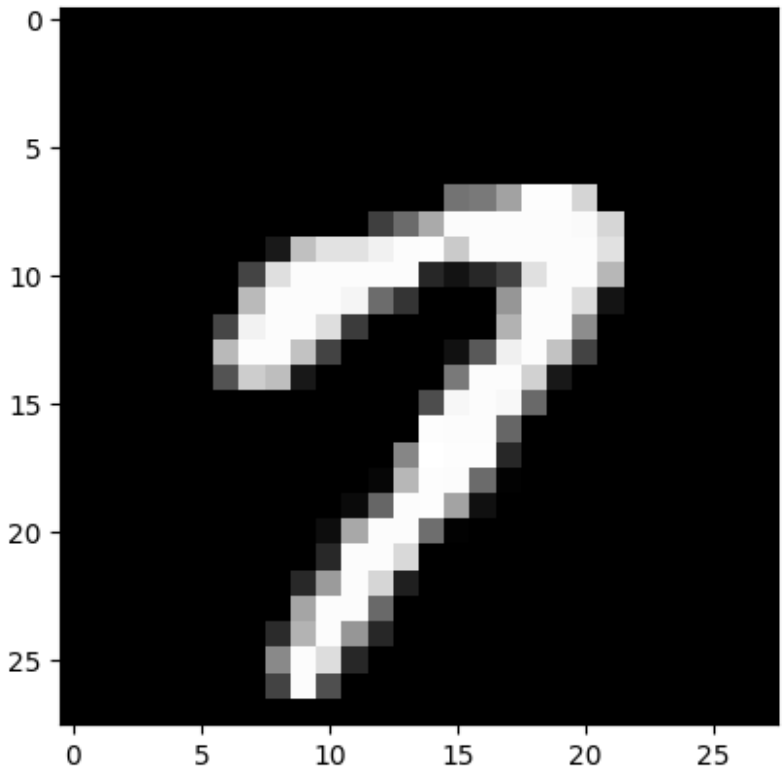
Explore the MNIST dataset:

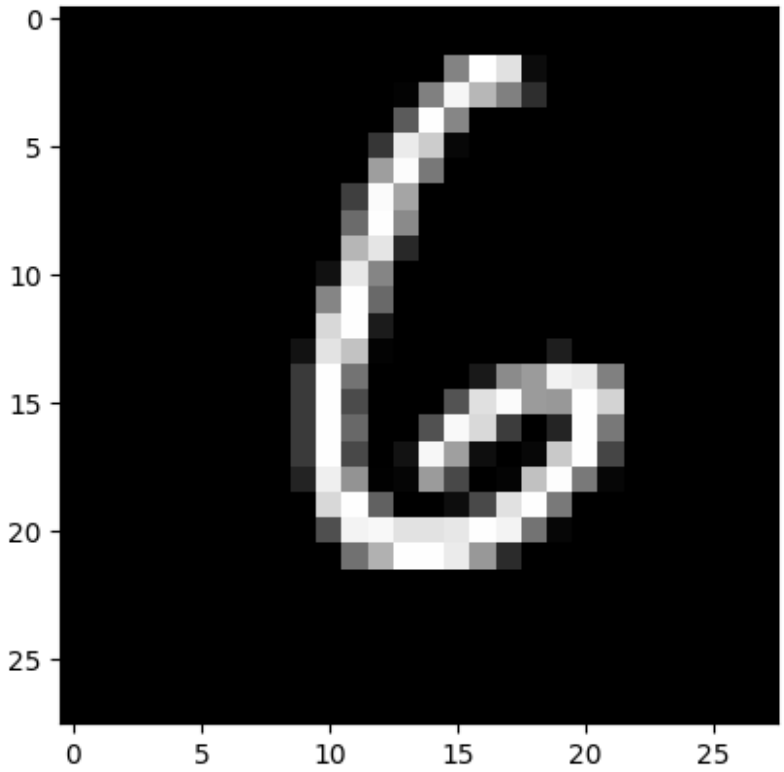
```
# TODO: Visualize samples of each class

classes, counts = np.unique(test_samples_labels, return_counts=True)

plt.bar(classes, counts)
plt.title("TEST SAMPLE CLASSES")
plt.xticks(range(0,10))
plt.xlabel("Classes")
plt.show()
```

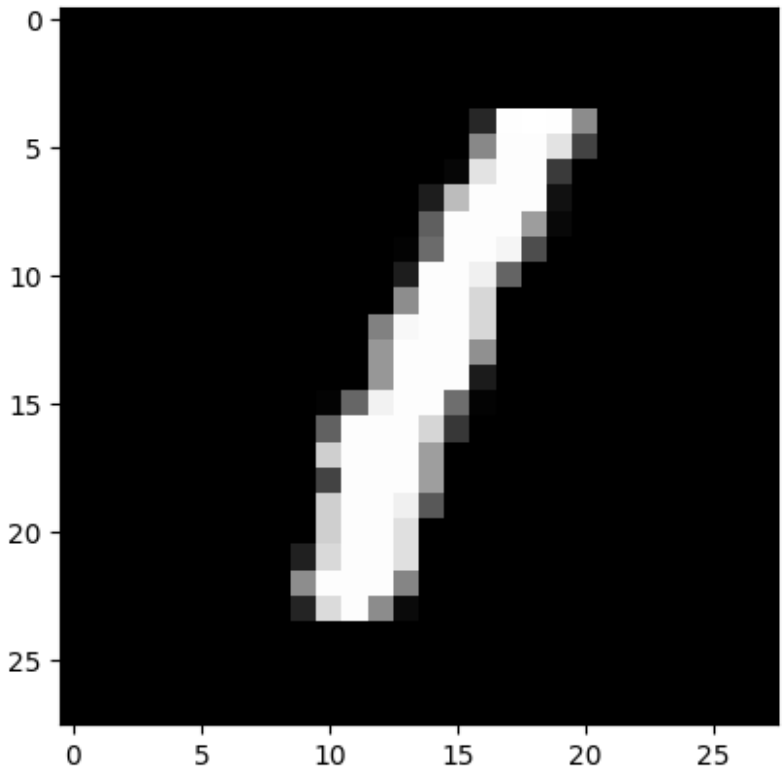
```
checklist = np.zeros(10)
for idx in range(len(train_samples)):
    if checklist[train_samples_labels[idx]] == 0:
        plt.imshow(train_samples[idx].reshape(28, 28), cmap='gray')
        plt.show()
        checklist[train_samples_labels[idx]] = 1
    elif ( checklist.all() == 1 ):
        break

# TODO: Display counts of each class
for c in classes:
    print(f"Class: {c} appears {counts[c]} times")
```

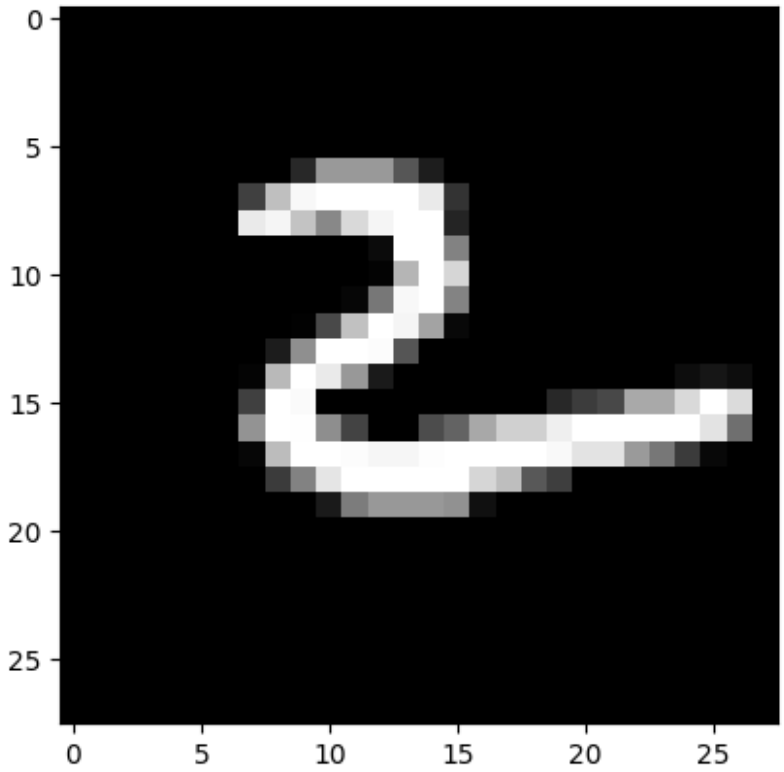


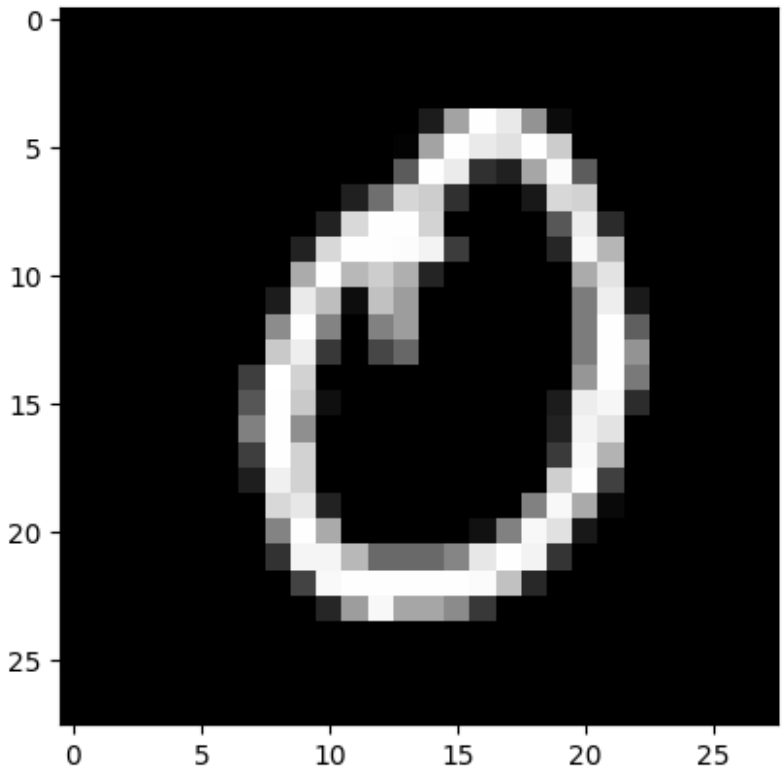

0
5
10
15
20
25
0
5
10
15
20
25

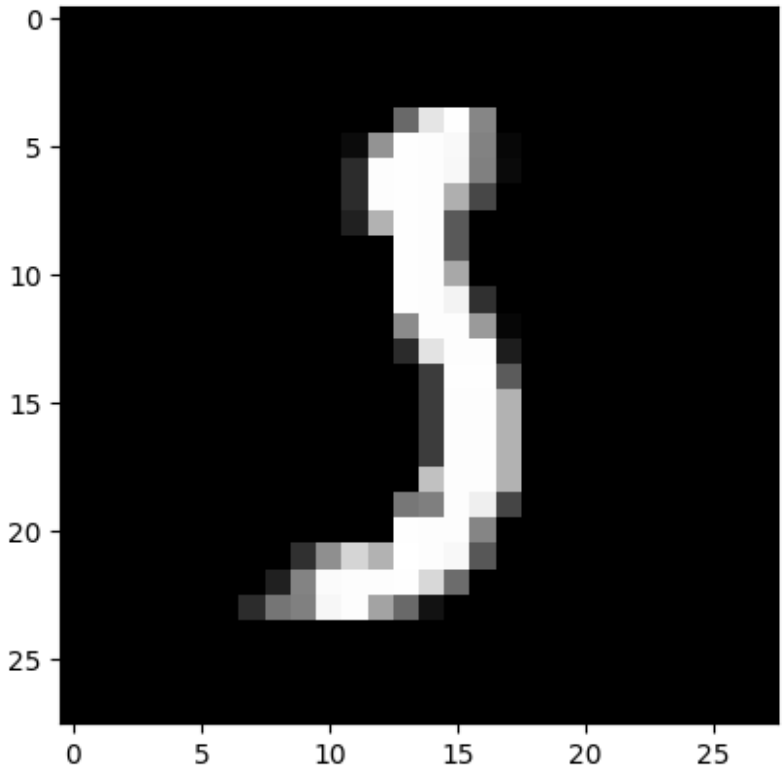

0
5
10
15
20
25
0
5
10
15
20
25

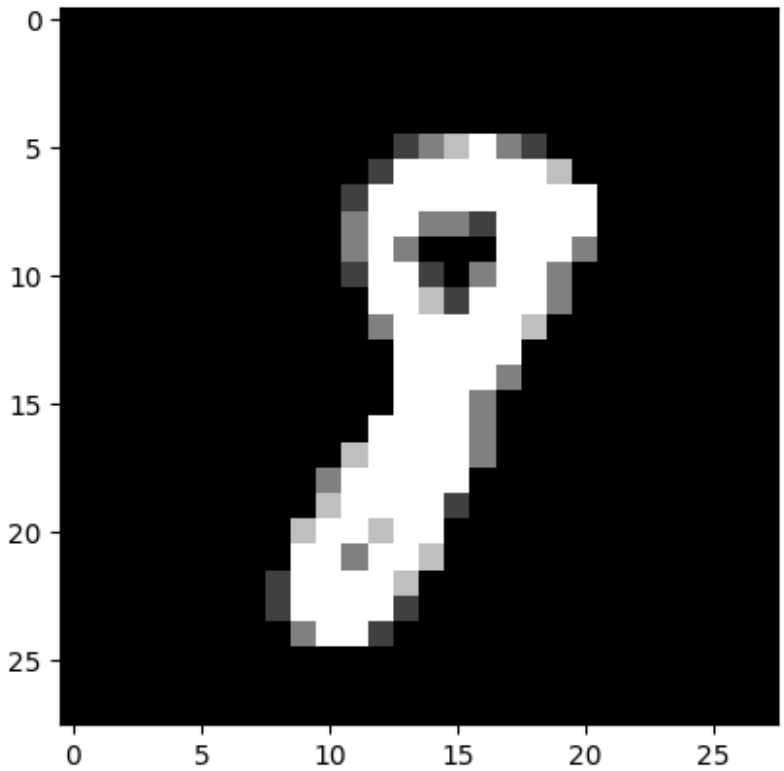

0
5
10
15
20
25
0
5
10
15
20
25

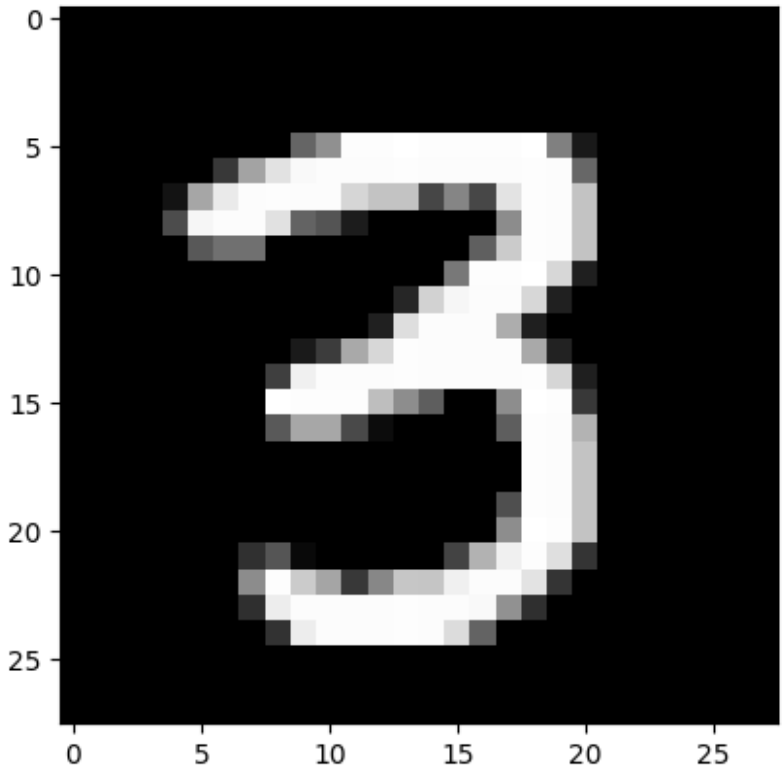

0
5
10
15
20
25
0
5
10
15
20
25

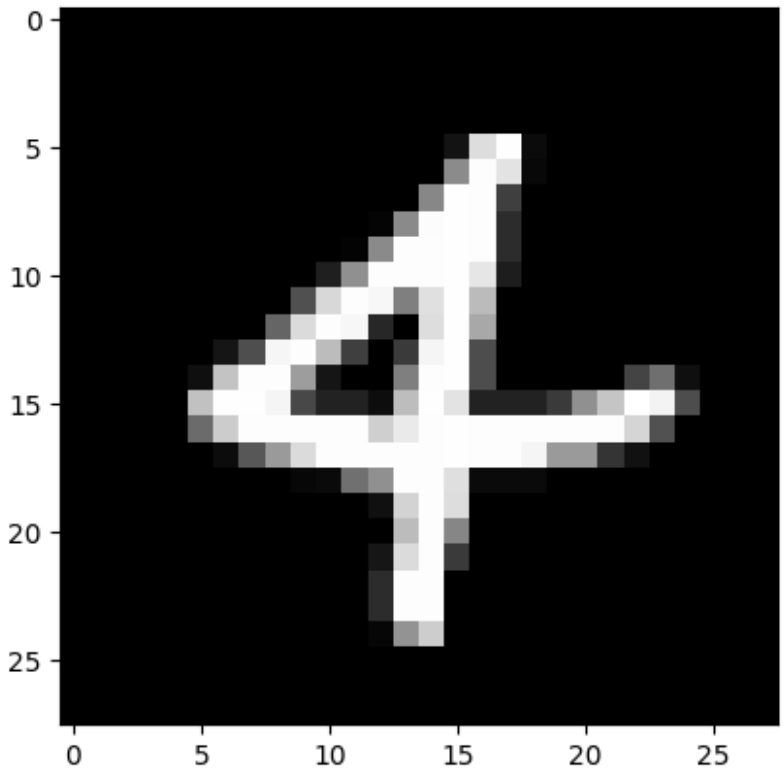

0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25






```
Class: 0 appears 86 times
Class: 1 appears 122 times
```

```
Class: 2 appears 113 times
Class: 3 appears 115 times
Class: 4 appears 108 times
Class: 5 appears 92 times
Class: 6 appears 87 times
Class: 7 appears 99 times
Class: 8 appears 86 times
Class: 9 appears 92 times
```

one-versus-the-rest (15 Points) and analysis

Using your implementation, compare multiclass classification performance of *one-versus-the-rest*

Create your own implementation of *one-versus-the-rest* and *one-versus-one*. Do not use sklearns multiclass SVM.

```
# TODO loop over classes training one_versus_the_rest()
# print(train_samples.shape)

trainBinaries = list()
testBinaries  = list()
for label in range(10):
    trainBinaries.append([1 if x == label else -1 for x in
train_samples_labels])
    testBinaries.append([1 if x == label else -1 for x in
test_samples_labels])


def getclassprobs(C=1.0):

    # Create the 10 one-vs-rest classifiers
    class_svms = []
    for label in range(10):

        svmForClass = SVC(C = C, kernel = 'linear', probability =
True)
        svmForClass.fit(train_samples, trainBinaries[label])
        class_svms.append(svmForClass)


    # TODO save all the prediction probability by predict_prob() for
the following function
    # Hint: svm = SVC(kernel="linear", probability=True)

        # For each classifier get the prediction that it IS that value
    classprobs_tr = list()
    classprobs_te = list()

    for label in range(10):
```

```
classprobs_tr.append(class_svms[label].predict_proba(train_samples)
[:,1])

classprobs_te.append(class_svms[label].predict_proba(test_samples)
[:,1])

    return np.array(classprobs_tr), np.array(classprobs_te),
class_svms

probs_tr, probs_te, class_svms = getclassprobs()
```

Determine the accuracy

```
def getaccuracies(classprob1, classprob2):
    result_tr = np.argmax(classprob1, axis=0)
    result_te = np.argmax(classprob2, axis=0)

    # print(result_te.shape)

    correct_tr = 0
    correct_te = 0
    for idx in range(len(result_tr)):
        if result_tr[idx] == train_samples_labels[idx]:
            correct_tr += 1
    for idx in range(len(result_te)):
        if result_te[idx] == test_samples_labels[idx]:
            correct_te += 1

    train_accuracy = correct_tr / len(result_tr)
    test_accuracy  = correct_te / len(result_te)

    return train_accuracy, test_accuracy

train_accuracy, test_accuracy = getaccuracies(probs_tr, probs_te)

# train_accuracy = np.nan #TODO
# test_accuracy  = np.nan #TODO

print("Train accuracy: {:.2f}".format(100*train_accuracy))
print("Test accuracy: {:.2f}".format(100*test_accuracy))
```

```
Train accuracy: 91.85
Test accuracy: 88.50
```

The parameter $C>0$ controls the tradeoff between the size of the margin and the slack variable penalty. It is analogous to the inverse of a regularization coefficient. Include in your report a brief discussion of how you found an appropriate value.

```
# Hint: Try using np.logspace for hyperparameter tuning
# TODO: Find an appropriate value of C.
train_accuracies = list()
test_accuracies = list()

# This part takes like 7 minutes.
c_vals = np.logspace(-5, 5, 11)
flatTrain = train_samples_labels.ravel()
flatTest  = test_samples_labels.ravel()

svm_list = list()

classprobs_tr_list = list()
classprobs_te_list = list()

# this part takes 11.5 minutes
for c in c_vals:

    classprobs_tr, classprobs_te, classprob_svm = getclassprobs(c)
    classprobs_tr_list.append(classprobs_tr)
    classprobs_te_list.append(classprobs_te)
    svm_list.append(classprob_svm)
    # svm_list.append(svm)

    tr_acc, te_acc = getaccuracies(classprobs_tr, classprobs_te)

    train_accuracies.append(tr_acc)
    test_accuracies.append(te_acc)
```

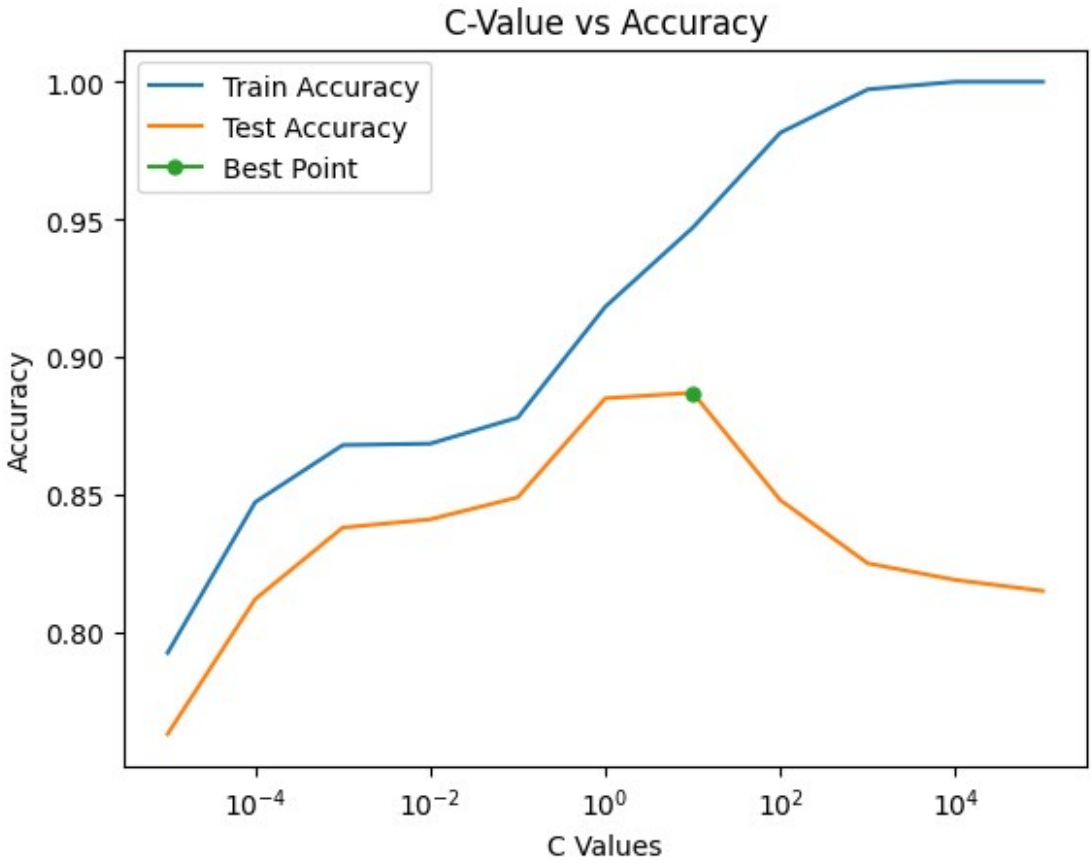
Provide details on how you found an appropriate value.

We wanted to get all the accuracies from all the C calues we tested. From there, we looked at all the accuracies and chose the C value that corresponded to the highest accuracy.

Plot accuracies for train and test using logspace for x-axis (i.e., C values)

```
# TODO: Plot the result.
bestCIdx = np.argmax(test_accuracies)
bestC = c_vals[bestCIdx]

plt.xscale('log')
plt.plot(c_vals, train_accuracies, label = 'Train Accuracy')
plt.plot(c_vals, test_accuracies, label = 'Test Accuracy')
plt.xlabel("C Values")
plt.ylabel("Accuracy")
plt.title("C-Value vs Accuracy")
plt.plot(bestC, test_accuracies[bestCIdx], label = "Best Point",
marker = "o", markersize = 5)
plt.legend()
```

```
<matplotlib.legend.Legend at 0x284533190>
```



What does this graph tell us about the importance of our C value?

TODO: Analyze the plot above:

The C value is another bias-variance tradeoff, as after a certain point (marked above by the green point), increasing the C value starts to decrease the accuracy of the model by making the data not generalize well to the new data. This can be seen by the drop in accuracy of the test accuracy after the green point. However, making the value too low makes the accuracy way too low, as it can't really generalize well to noise. This shows that the C value is also a way to tune over and under fitting.

(10 Points)

In addition to calculating percent accuracy, generate multiclass confusion matrices as part of your analysis.

```
train_predictions = list()
test_predictions = list()
# TODO
result_tr = np.argmax(probs_tr, axis=0)
result_te = np.argmax(probs_te, axis=0)
```

```
# cm_tr = confusion_matrix(train_samples_labels, result_tr)
# cm_te = confusion_matrix(test_samples_labels,  result_te)

# svm = SVC(C = bestC, kernel = 'linear')
# svm.fit(train_samples, flatTrain)

# train_predictions = svm.predict(train_samples)
# test_predictions  = svm.predict(test_samples)

cm_tr = confusion_matrix(train_samples_labels, result_tr)
cm_te = confusion_matrix(test_samples_labels, result_te)
print("TRAIN MATRIX")
print(cm_tr)
print("TEST MATRIX")
print(cm_te)

# for label in range(10):
#     print("Label:", label)
#     # cm_tr = confusion_matrix(trainBinaries[label],
svm_list[bestCIdx][label].predict(train_samples))
#     cm_te = confusion_matrix(testBinaries[label], svm_list[bestCIdx]
[label].predict(test_samples))
#     # print(" Train:")
#     # print(cm_tr)
#     # print(" Test:")
#     display(pd.DataFrame(cm_te))

TRAIN MATRIX
[[374   0   1   0   0   1   0   0   4   2]
 [  0 434   3   0   0   5   1   0   7   1]
 [  2   4 368   4   7   0   7   4  10   4]
 [  0   1  12 371   0   6   1   7  10   5]
 [  0   2   4   0 386   0   1   3   2  19]
 [  4   4   2   9   3 299  12   1   9   4]
 [  1   2   1   0   1   6 383   0   0   0]
 [  0   1   3   1   8   0   0 388   0  11]
 [  2   6  11  10   2   8   4   2 319   9]
 [  2   0   4   9  14   4   0  12   4 352]]
TEST MATRIX
[[ 84   0   0   0   0   1   1   0   0   0]
 [  0 120   0   1   0   0   0   0   1   0]
 [  0   0  99   2   0   0   1   3   6   2]
 [  0   0   1  91   0  11   4   4   3   1]
 [  0   0   0   0  95   1   5   0   1   6]
 [  1   0   1   3   0  81   2   0   4   0]
 [  2   0   0   0   0   4  79   1   1   0]
 [  1   2   5   0   2   0   0  87   0   2]
 [  0   0   2   4   5   4   0   2  65   4]
 [  0   1   0   1   2   1   0   1   2  84]]
```

Evaluation (15 points)

Now we will report our results and compare to other algorithms. Usually compare with a handful Logisitic regression

Create your own implementation of *one-versus-the-rest* and *one-versus-one*. Do not use sklearns multiclass Logistic Regression.

```
train_predictions = list()
test_predictions = list()

# Creates the aggreagated binary classifiers for each class for train
and test


# Create all the models for all of the different C values
logList = list() # The list with all of the predictors
for c in c_vals: # For each C value

    cPredictor = list() # The list that has the 10 classifiers for the
corresponding C
    for label in range(10):
        logRegression = LogisticRegression(C = c)
        logRegression.fit(train_samples, trainBinaries[label])
        cPredictor.append(logRegression)
    logList.append(cPredictor)


# TODO
# for classifier in svm_list:
#     pred_tr = classifier.predict(train_samples)
#     pred_te = classifier.predict(test_samples)

#     train_predictions.append(pred_tr)
#     test_predictions.append(pred_te)
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
```

```
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
```

```
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
```

```
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
```

```
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
```

```
https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
```

```
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
```

```
https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
# Get the predictions for each C value (NOTE: SEPARATED FOR TESTING
PURPOSES)
for predictor in logList:
    trainProba = np.array([model.predict_proba(train_samples) for
model in predictor])
    testProba  = np.array([model.predict_proba(test_samples) for model
in predictor])

    train_predictions.append(np.argmax(trainProba, axis = 0))
    test_predictions.append(np.argmax(testProba, axis = 0))
```

Create a table comparing model accuracy on train and test data.

```
# # TODO
# Get the actual accuracies
logTrainAccuracies = [len(prediction[prediction ==
train_samples_labels]) / len(train_samples_labels) for prediction in
train_predictions]
logTestAccuracies  = [len(prediction[prediction ==
test_samples_labels])  / len(test_samples_labels)  for prediction in
test_predictions ]

bestLogCIdx = np.argmax(logTrainAccuracies)

# DF Data
compareResults = {"Logistic": [logTrainAccuracies[bestLogCIdx],
logTestAccuracies[bestLogCIdx]],
                  "SVM": [train_accuracies[bestCIdx],
test_accuracies[bestCIdx]]}

df = pd.DataFrame(compareResults, index = ["Train", "Test"])
display(df)
```

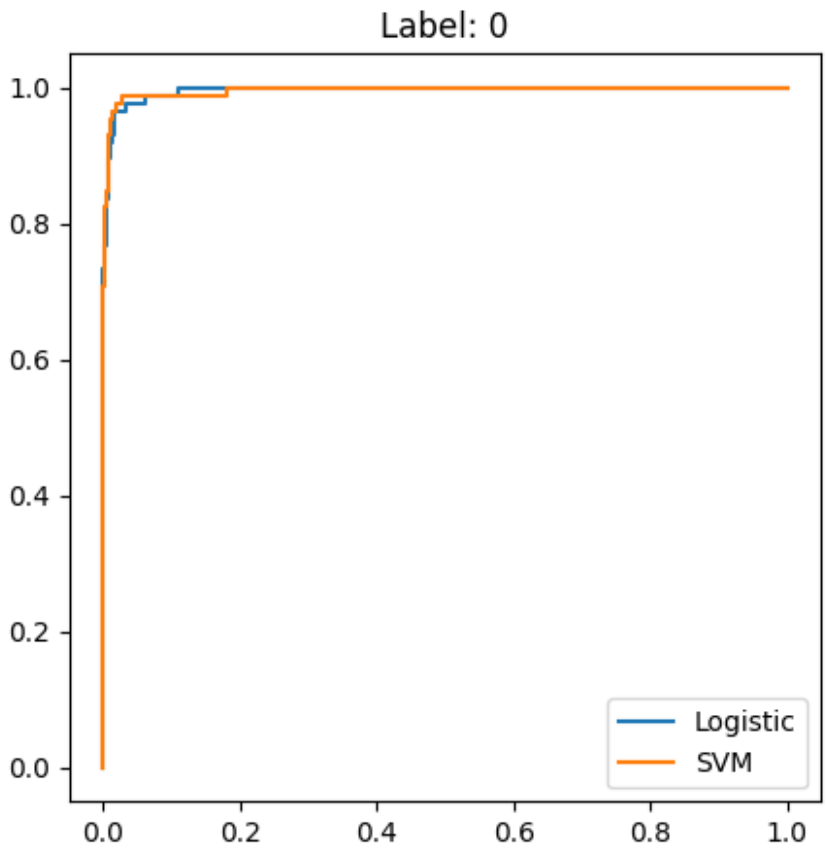
```
# acc_dict = dict()
# tr_acc_list = list()
# te_acc_list = list()
# new_svm_list = list()
# tr_binaries, te_binaries = list(), list()

# for c in classes:
#     tr_samples_binary, te_samples_binary = list(), list()
#     for idx in range(len(train_samples_labels)):
#         tr_samples_binary.append( 1 if (train_samples_labels[idx] ==
c) else -1 )
#     for idx in range(len(test_samples_labels)):
#         te_samples_binary.append( 1 if (test_samples_labels[idx] ==
c) else -1 )

#     tr_binaries.append(tr_samples_binary)
#     te_binaries.append(te_samples_binary)

#     svm = SVC(kernel="linear", C=bestC)
#     svm.fit(train_samples, tr_samples_binary)

#     new_svm_list.append(svm)

# for idx in range(len(new_svm_list)):
#     tr_acc_list.append(new_svm_list[idx].score(train_samples,
tr_binaries[idx]))
#     te_acc_list.append(new_svm_list[idx].score(test_samples,
te_binaries[idx]))
```

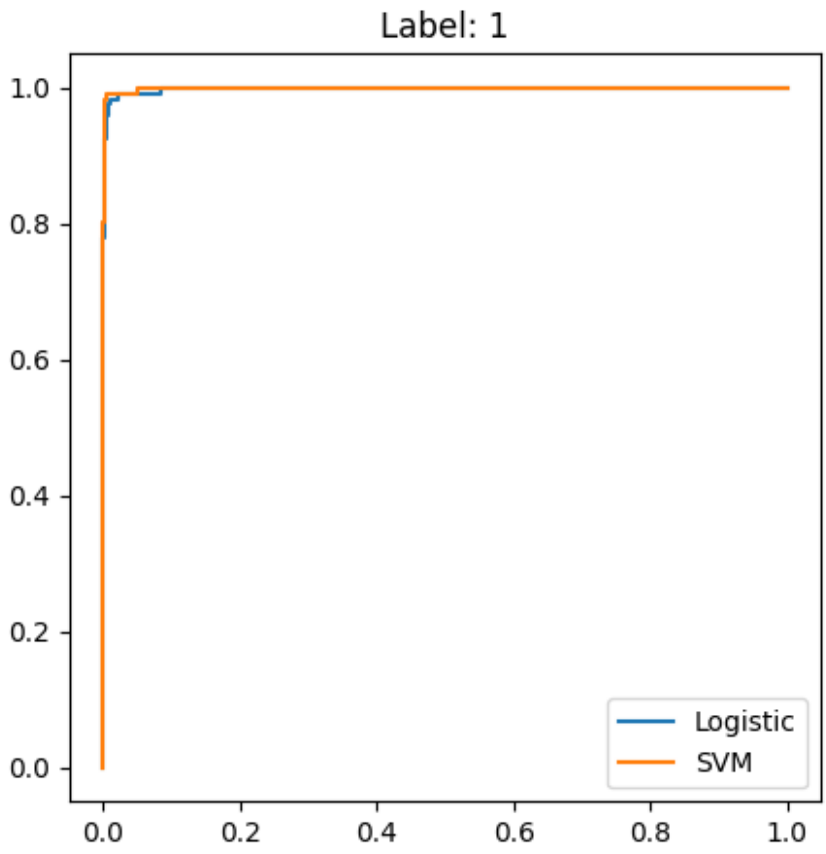
```
       Logistic    SVM
Train   0.99775  0.947
Test    0.82200  0.887
```

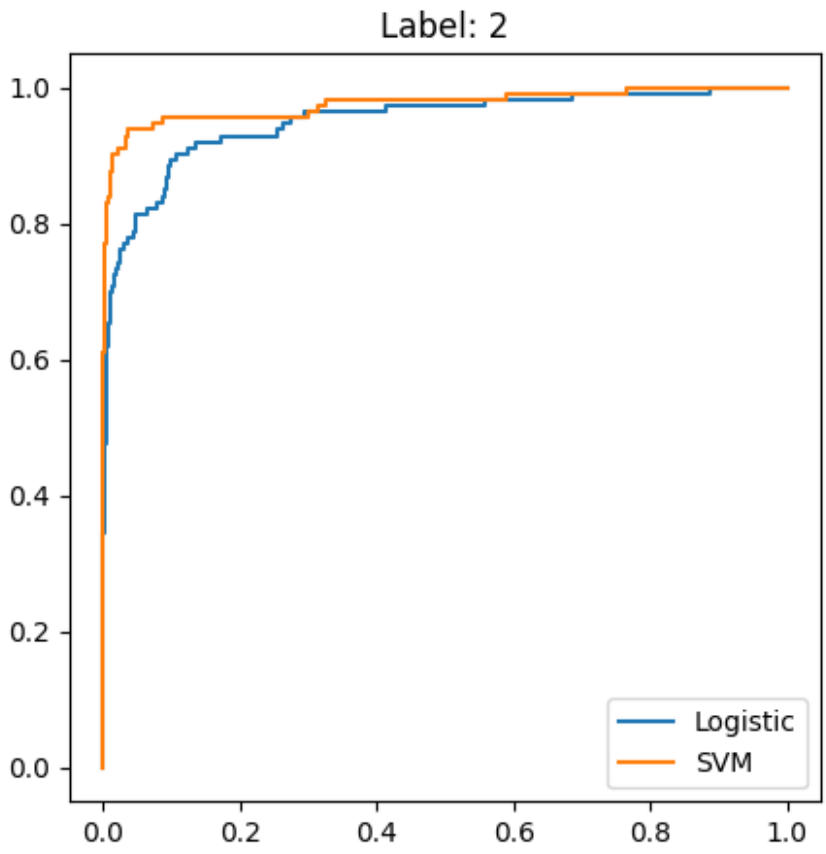
Create 9 graphs (one for each label) with two ROC curves (one for each model).

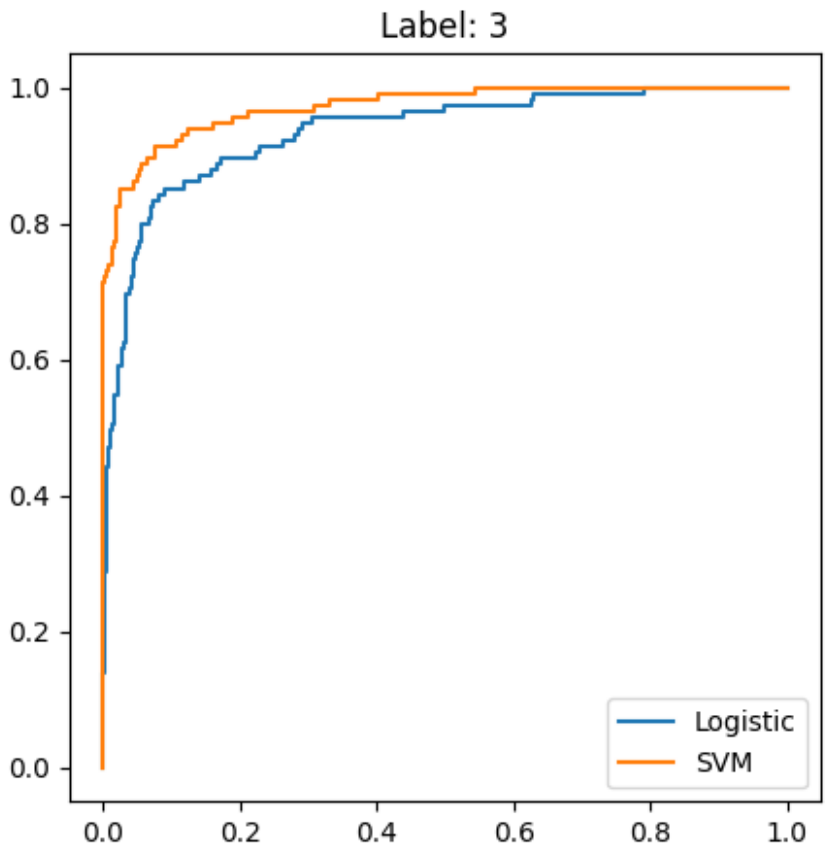
```
# TODO
from sklearn.metrics import roc_curve


for label in range(10):
    logFpr, logTpr, logThr = roc_curve(testBinaries[label],
logList[bestLogCIdx][label].predict_proba(test_samples)[:, 1])

    svmFpr, svmTpr, svmThr = roc_curve(testBinaries[label],
svm_list[bestCIdx][label].predict_proba(test_samples)[:, 1])
    plt.figure(figsize = (5, 5))
    plt.title("Label: " + str(label))
    plt.plot(logFpr, logTpr, label = "Logistic")
    plt.plot(svmFpr, svmTpr, label = "SVM")
    plt.xlabel = "False Positive Rate"
```

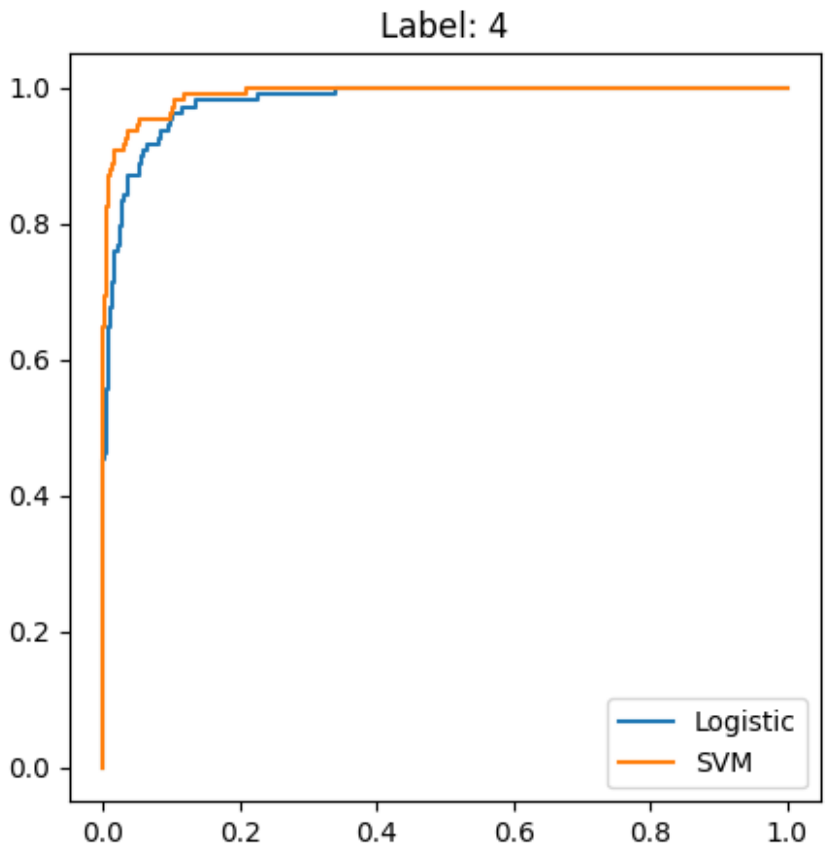
```
    plt.ylabel = "True Positive Rate"
    plt.legend(loc='lower right')
    plt.show()



# logDataPoints = np.array(logDataPoints)
# svmDataPoints = np.array(svmDataPoints)
# print(logDataPoints[0, 0, 0])
# subplotVal = 111
# for label in range(10):
#     plt.subplot(subplotVal + label)
#     plt.plot(logDataPoints[label, :, 0], logDataPoints[label, :, 1])
#     plt.plot(svmDataPoints[label, :, 0], svmDataPoints[label, :, 1])

# logDataPoints = list()
# svmDataPoints = list()
# for label in range(10):
#     logLabelList = list()
#     svmLabelList = list()

#     for cIdx in range(len(c_vals)):
#         c = c_vals[cIdx]

#         logFpr, logTpr, _  = roc_curve(testBinaries[label],
logList[cIdx][label].predict(test_samples))
#         logLabelList.append((logFpr[1], logTpr[1]))

#         svmBinary = [1 if prediction == label else -1 for prediction
in svm_list[cIdx].predict(test_samples)]
#         svmFpr, svmTpr, _ = roc_curve(testBinaries[label],
svmBinary)
#         svmLabelList.append((svmFpr[1], svmTpr[1]))

#     logDataPoints.append(logLabelList)
#     svmDataPoints.append(svmLabelList)

# for idx in range(len(new_svm_list)):
#     fpr, tpr, thr = roc_curve(tr_binaries[idx], \
#
new_svm_list[idx].predict(train_samples))
#     plt.plot(fpr, tpr, label=f"ROC {idx}")

# plt.xlabel = "False Positive Rate"
# plt.ylabel = "True Positive Rate"
# plt.legend(loc='lower right')
# plt.show()
```

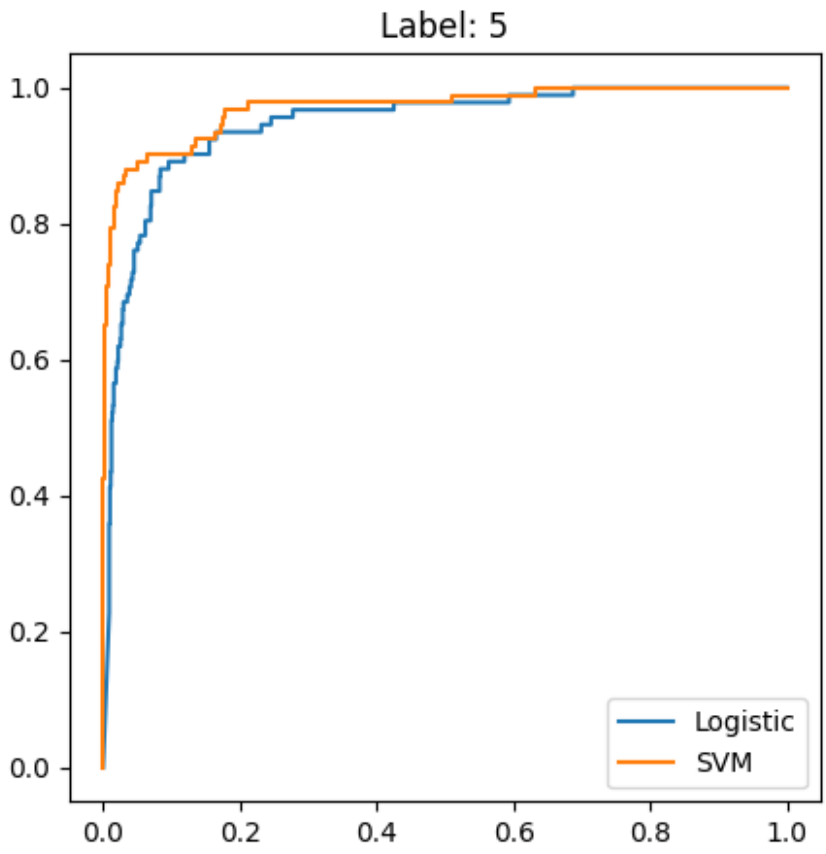

Label: 0
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Logistic
SVM

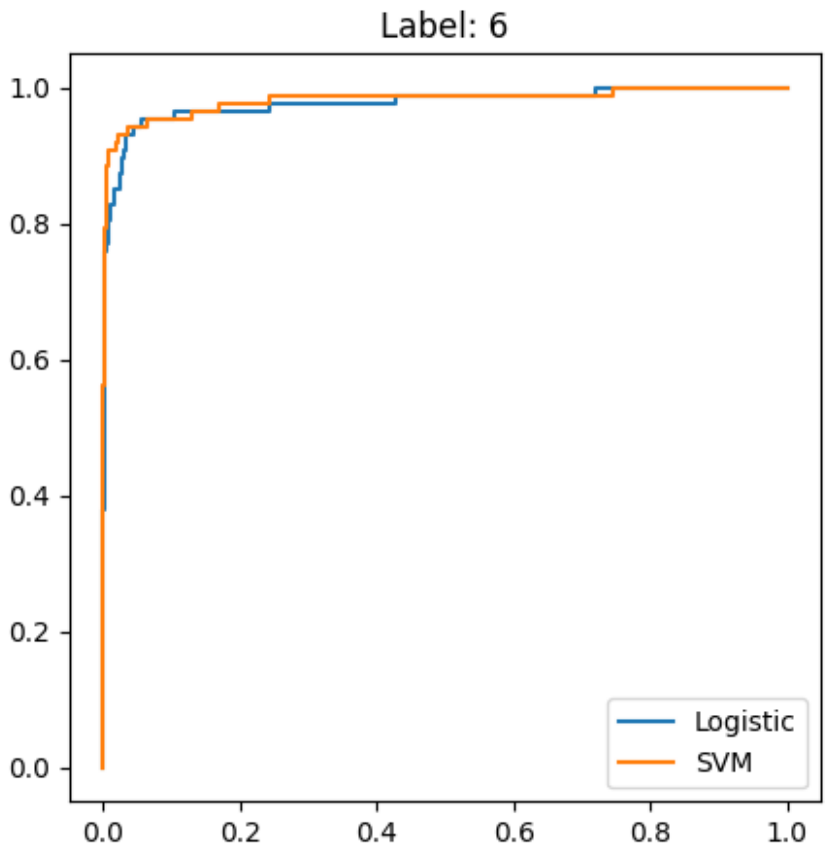

Label: 1
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Logistic
SVM

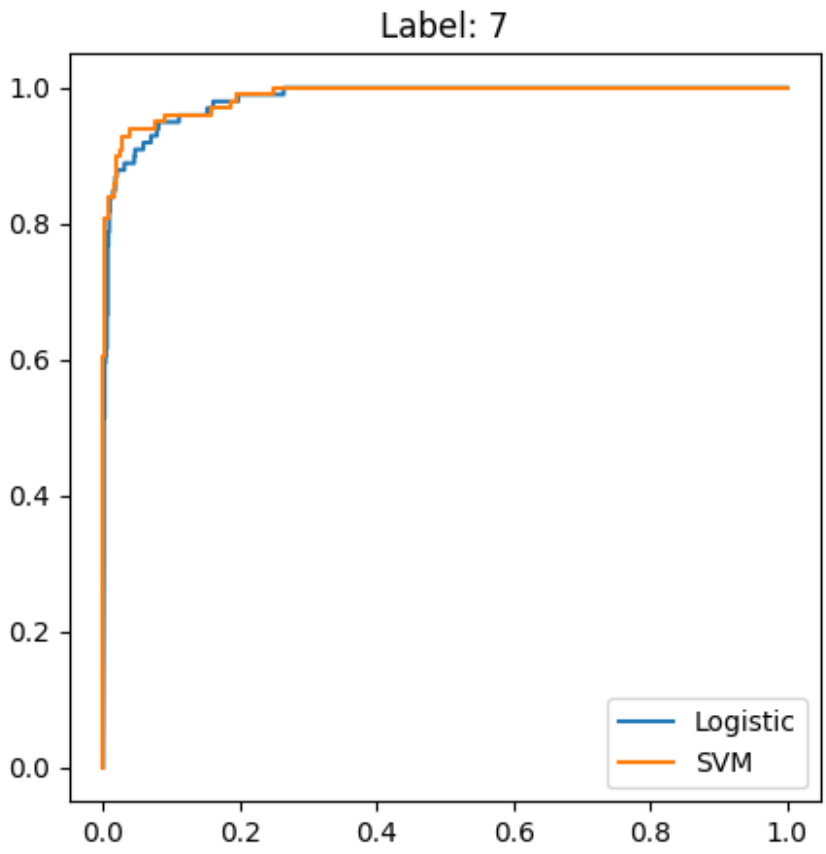

Label: 2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Logistic
SVM

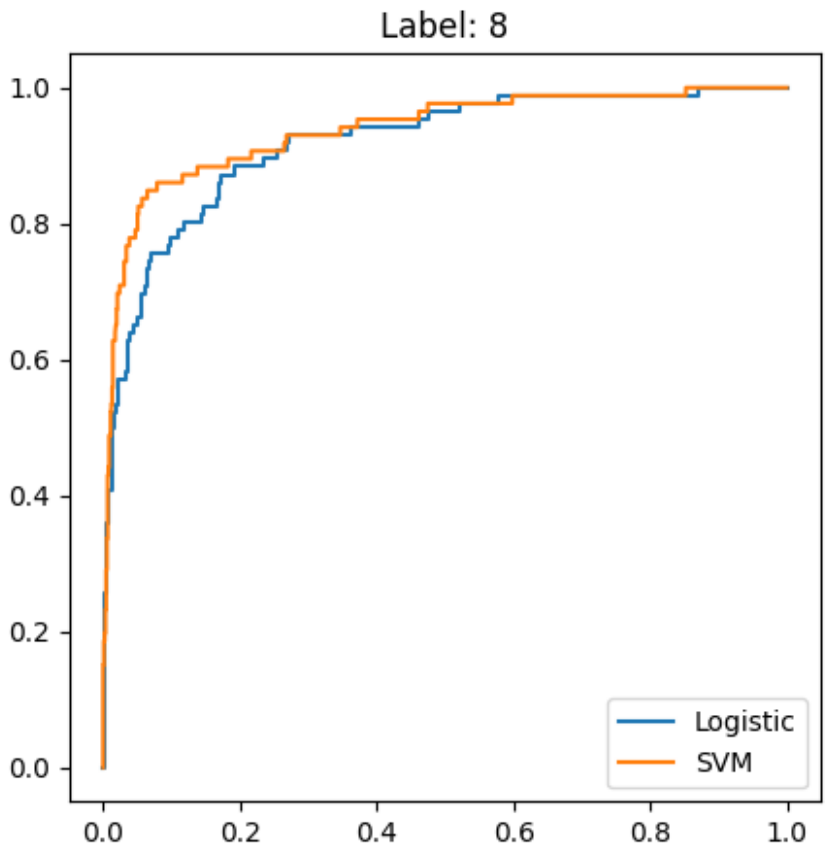

Label: 3
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Logistic
SVM

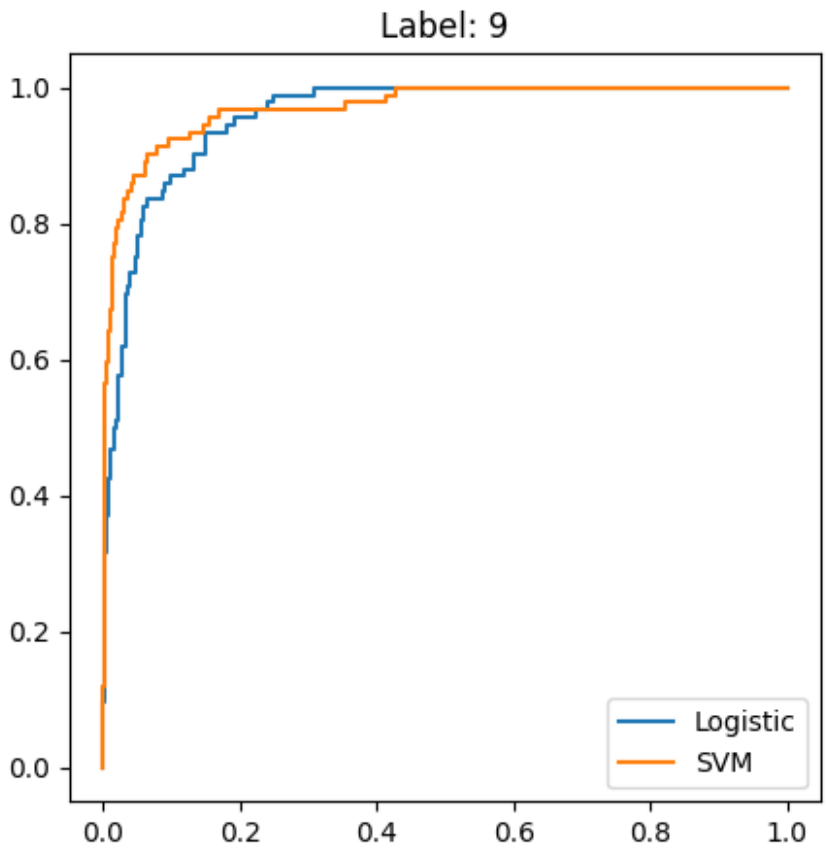

Label: 4
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Logistic
SVM


Label: 5
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Logistic
SVM


Label: 6
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Logistic
SVM


Label: 7
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Logistic
SVM


Label: 8
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Logistic
SVM



BONUS (+5 points): Non-linear kernel

Intuition Behind Kernels

The SVM classifier obtained by solving the convex Lagrange dual of the primal max-margin SVM formulation is as follows:

$$f(x)=\sum_{i=1}^{N} \alpha_i \cdot y_i \cdot K(x, x_i)+b,$$

where N is the number of support vectors.

If you know the intuition behind a linear discriminant function, the non-parametric SVM classifier above is very easy to understand. Instead of imagining the original features of each data point, consider a transformation to a new feature space where the data point has N features, one for each support vector. The value of the i^{th} feature is equal to the value of the kernel between the i^{th} support vector and the data point is classified. The original (possibly non-linear) SVM classifier is like any other linear discriminant in this space.

Note that after the transformation, the original features of the data point are irrelevant. Its dot products with support vectors (special data points chosen by the SVM optimization algorithm) represent it only. One of my professors used a loose analogy while explaining

this idea: A person has seen lakes, rivers, streams, fords, etc., but has never seen the sea. How would you explain to this person what a sea is? By relating the amount of water in an ocean to that found in a water body, the person already knows, etc.

In some instances, like the RBF kernel, defining the transformed features in terms of the original features of a data point leads to an infinite-dimensional representation. Unfortunately, though this an awe-inspiring fact often mentioned while explaining how powerful SVMs are, it drops in only after repeated encounters with the idea ranging from introductory machine learning to statistical learning theory.

Intuition Behind Gaussian Kernels

The Gaussian/RBF kernel is as follows:

$$K(x, y) = exp\left(-\frac{|)x - y|)^2}{2\sigma^2}\right)$$

Like any other kernel, we can understand the RBF kernel regarding feature transformation via the dot products given above. However, the intuition that helps best when analyzing the RBF kernel is that of the Gaussian distribution (as provided by Akihiro Matsukawa).

The Gaussian kernel computed with a support vector is an exponentially decaying function in the input feature space, the maximum value of which is attained at the support vector and which decays uniformly in all directions around the support vector, leading to hyper-spherical contours of the kernel function. The SVM classifier with the Gaussian kernel is simply a weighted linear combination of the kernel function computed between a data point and each support vector. The role of a support vector in the classification of a data point gets tempered with α, the global prediction usefulness of the support vector, and $K(x, y)$, the local influence of a support vector in prediction at a particular data point.

In the 2D feature space, each support vector's kernel function's heat map decay away from the support vector and the resulting classifier (see the following figure).

Notion of Universal Kernels

(This comes from learning theory, it could be more intuitive, but good to know.)

Gaussian kernels are universal kernels, i.e., their use with appropriate regularization guarantees a globally optimal predictor, which minimizes a classifier's estimation and approximation errors. Here, we incur approximation error by limiting the space of classification models over which the search space. Estimation error refers to errors in estimating the model parameters.

Intuition Behind Gaussian Kernels

The Gaussian/RBF kernel is as follows:

$$K(x, y) = exp\left(-\frac{||x - y||^2}{2\sigma^2}\right)$$

Like any other kernel, we can understand the RBF kernel regarding feature transformation via the dot products given above. However, the intuition that helps best when analyzing the RBF kernel is that of the Gaussian distribution (as provided by Akihiro Matsukawa).

The Gaussian kernel computed with a support vector is an exponentially decaying function in the input feature space, the maximum value of which is attained at the support vector and which decays uniformly in all directions around the support vector, leading to hyper-spherical contours of the kernel function. The SVM classifier with the Gaussian kernel is simply a weighted linear combination of the kernel function computed between a data point and each support vector. The role of a support vector in the classification of a data point gets tempered with α, the global prediction usefulness of the support vector, and $K(x, y)$, the local influence of a support vector in prediction at a particular data point.

In the 2D feature space, each support vector's kernel function's heat map decay away from the support vector and the resulting classifier (see the following figure).

[gkernel in 2D](images/gkernel-2d.jpeg " kernel function of each support vector")

Notion of Universal Kernels

(This comes from learning theory, it could be more intuitive, but good to know.)

Gaussian kernels are universal kernels, i.e., their use with appropriate regularization guarantees a globally optimal predictor, which minimizes a classifier's estimation and approximation errors. Here, we incur approximation error by limiting the space of classification models over which the search space. Estimation error refers to errors in estimating the model parameters.

Implement `nonlinear_kernel()` in `implementation.py`, use it, and compare with others (repeat above for SVM using non-linear kernel and do analysis).

```
# (Bonus) TODO
nonlin_ref = SVC(kernel="rbf")
nonlin_ref.fit(X, y)

nonlin_svm = SVM(kernel=nonlinear_kernel)
nonlin_svm.fit(X, y)

print()
print("Alpha", nonlin_svm.a)
print("Weights", nonlin_svm.w)
print("Bias", nonlin_svm.b)
```

```
[[ 1 -2  4]
 [ 1  4  1]
 [ 1  1  6]
 [ 1  2  4]
```

```
 [ 1  6  2]]
[[ 0. 45. 13. 16. 68.]
 [45.  0. 34. 13.  5.]
 [13. 34.  0.  5. 41.]
 [16. 13.  5.  0. 20.]
 [68.  5. 41. 20.  0.]]
[[1.00000000e+00 8.19401262e-40 5.10908903e-12 1.26641655e-14
  8.62880116e-60]
 [8.19401262e-40 1.00000000e+00 2.93748211e-30 5.10908903e-12
  4.53999298e-05]
 [5.10908903e-12 2.93748211e-30 1.00000000e+00 4.53999298e-05
  2.44260074e-36]
 [1.26641655e-14 5.10908903e-12 4.53999298e-05 1.00000000e+00
  4.24835426e-18]
 [8.62880116e-60 4.53999298e-05 2.44260074e-36 4.24835426e-18
  1.00000000e+00]]
[[ 1 -2  4]
 [ 1  4  1]
 [ 1  1  6]
 [ 1  2  4]
 [ 1  6  2]]
[[ 0. 45. 13. 16. 68.]
 [45.  0. 34. 13.  5.]
 [13. 34.  0.  5. 41.]
 [16. 13.  5.  0. 20.]
 [68.  5. 41. 20.  0.]]
[[1.00000000e+00 8.19401262e-40 5.10908903e-12 1.26641655e-14
  8.62880116e-60]
 [8.19401262e-40 1.00000000e+00 2.93748211e-30 5.10908903e-12
  4.53999298e-05]
 [5.10908903e-12 2.93748211e-30 1.00000000e+00 4.53999298e-05
  2.44260074e-36]
 [1.26641655e-14 5.10908903e-12 4.53999298e-05 1.00000000e+00
  4.24835426e-18]
 [8.62880116e-60 4.53999298e-05 2.44260074e-36 4.24835426e-18
  1.00000000e+00]]
[[ 1 -2  4]
 [ 1  4  1]
 [ 1  1  6]
 [ 1  2  4]
 [ 1  6  2]]
[[ 0. 45. 13. 16. 68.]
 [45.  0. 34. 13.  5.]
 [13. 34.  0.  5. 41.]
 [16. 13.  5.  0. 20.]
 [68.  5. 41. 20.  0.]]
[[1.00000000e+00 8.19401262e-40 5.10908903e-12 1.26641655e-14
  8.62880116e-60]
 [8.19401262e-40 1.00000000e+00 2.93748211e-30 5.10908903e-12
  4.53999298e-05]
```

```
 [5.10908903e-12 2.93748211e-30 1.00000000e+00 4.53999298e-05
  2.44260074e-36]
 [1.26641655e-14 5.10908903e-12 4.53999298e-05 1.00000000e+00
  4.24835426e-18]
 [8.62880116e-60 4.53999298e-05 2.44260074e-36 4.24835426e-18
  1.00000000e+00]]
[[ 1 -2  4]
 [ 1  4  1]
 [ 1  1  6]
 [ 1  2  4]
 [ 1  6  2]]
[[ 0. 45. 13. 16. 68.]
 [45.  0. 34. 13.  5.]
 [13. 34.  0.  5. 41.]
 [16. 13.  5.  0. 20.]
 [68.  5. 41. 20.  0.]]
[[1.00000000e+00 8.19401262e-40 5.10908903e-12 1.26641655e-14
  8.62880116e-60]
 [8.19401262e-40 1.00000000e+00 2.93748211e-30 5.10908903e-12
  4.53999298e-05]
 [5.10908903e-12 2.93748211e-30 1.00000000e+00 4.53999298e-05
  2.44260074e-36]
 [1.26641655e-14 5.10908903e-12 4.53999298e-05 1.00000000e+00
  4.24835426e-18]
 [8.62880116e-60 4.53999298e-05 2.44260074e-36 4.24835426e-18
  1.00000000e+00]]
[[ 1 -2  4]
 [ 1  4  1]
 [ 1  1  6]
 [ 1  2  4]
 [ 1  6  2]]
[[ 0. 45. 13. 16. 68.]
 [45.  0. 34. 13.  5.]
 [13. 34.  0.  5. 41.]
 [16. 13.  5.  0. 20.]
 [68.  5. 41. 20.  0.]]
[[1.00000000e+00 8.19401262e-40 5.10908903e-12 1.26641655e-14
  8.62880116e-60]
 [8.19401262e-40 1.00000000e+00 2.93748211e-30 5.10908903e-12
  4.53999298e-05]
 [5.10908903e-12 2.93748211e-30 1.00000000e+00 4.53999298e-05
  2.44260074e-36]
 [1.26641655e-14 5.10908903e-12 4.53999298e-05 1.00000000e+00
  4.24835426e-18]
 [8.62880116e-60 4.53999298e-05 2.44260074e-36 4.24835426e-18
  1.00000000e+00]]
[[ 1 -2  4]
 [ 1  4  1]
 [ 1  1  6]
 [ 1  2  4]
```

```
 [ 1  6  2]]
[[ 0. 45. 13. 16. 68.]
 [45.  0. 34. 13.  5.]
 [13. 34.  0.  5. 41.]
 [16. 13.  5.  0. 20.]
 [68.  5. 41. 20.  0.]]
[[1.00000000e+00 8.19401262e-40 5.10908903e-12 1.26641655e-14
  8.62880116e-60]
 [8.19401262e-40 1.00000000e+00 2.93748211e-30 5.10908903e-12
  4.53999298e-05]
 [5.10908903e-12 2.93748211e-30 1.00000000e+00 4.53999298e-05
  2.44260074e-36]
 [1.26641655e-14 5.10908903e-12 4.53999298e-05 1.00000000e+00
  4.24835426e-18]
 [8.62880116e-60 4.53999298e-05 2.44260074e-36 4.24835426e-18
  1.00000000e+00]]
[[ 1 -2  4]
 [ 1  4  1]
 [ 1  1  6]
 [ 1  2  4]
 [ 1  6  2]]
[[ 0. 45. 13. 16. 68.]
 [45.  0. 34. 13.  5.]
 [13. 34.  0.  5. 41.]
 [16. 13.  5.  0. 20.]
 [68.  5. 41. 20.  0.]]
[[1.00000000e+00 8.19401262e-40 5.10908903e-12 1.26641655e-14
  8.62880116e-60]
 [8.19401262e-40 1.00000000e+00 2.93748211e-30 5.10908903e-12
  4.53999298e-05]
 [5.10908903e-12 2.93748211e-30 1.00000000e+00 4.53999298e-05
  2.44260074e-36]
 [1.26641655e-14 5.10908903e-12 4.53999298e-05 1.00000000e+00
  4.24835426e-18]
 [8.62880116e-60 4.53999298e-05 2.44260074e-36 4.24835426e-18
  1.00000000e+00]]
[[ 1 -2  4]
 [ 1  4  1]
 [ 1  1  6]
 [ 1  2  4]
 [ 1  6  2]]
[[ 0. 45. 13. 16. 68.]
 [45.  0. 34. 13.  5.]
 [13. 34.  0.  5. 41.]
 [16. 13.  5.  0. 20.]
 [68.  5. 41. 20.  0.]]
[[1.00000000e+00 8.19401262e-40 5.10908903e-12 1.26641655e-14
  8.62880116e-60]
 [8.19401262e-40 1.00000000e+00 2.93748211e-30 5.10908903e-12
  4.53999298e-05]
```

```
 [5.10908903e-12 2.93748211e-30 1.00000000e+00 4.53999298e-05
  2.44260074e-36]
 [1.26641655e-14 5.10908903e-12 4.53999298e-05 1.00000000e+00
  4.24835426e-18]
 [8.62880116e-60 4.53999298e-05 2.44260074e-36 4.24835426e-18
  1.00000000e+00]]
[[ 1 -2  4]
 [ 1  4  1]
 [ 1  1  6]
 [ 1  2  4]
 [ 1  6  2]]
[[ 0. 45. 13. 16. 68.]
 [45.  0. 34. 13.  5.]
 [13. 34.  0.  5. 41.]
 [16. 13.  5.  0. 20.]
 [68.  5. 41. 20.  0.]]
[[1.00000000e+00 8.19401262e-40 5.10908903e-12 1.26641655e-14
  8.62880116e-60]
 [8.19401262e-40 1.00000000e+00 2.93748211e-30 5.10908903e-12
  4.53999298e-05]
 [5.10908903e-12 2.93748211e-30 1.00000000e+00 4.53999298e-05
  2.44260074e-36]
 [1.26641655e-14 5.10908903e-12 4.53999298e-05 1.00000000e+00
  4.24835426e-18]
 [8.62880116e-60 4.53999298e-05 2.44260074e-36 4.24835426e-18
  1.00000000e+00]]
[[ 1 -2  4]
 [ 1  4  1]
 [ 1  1  6]
 [ 1  2  4]
 [ 1  6  2]]
[[ 0. 45. 13. 16. 68.]
 [45.  0. 34. 13.  5.]
 [13. 34.  0.  5. 41.]
 [16. 13.  5.  0. 20.]
 [68.  5. 41. 20.  0.]]
[[1.00000000e+00 8.19401262e-40 5.10908903e-12 1.26641655e-14
  8.62880116e-60]
 [8.19401262e-40 1.00000000e+00 2.93748211e-30 5.10908903e-12
  4.53999298e-05]
 [5.10908903e-12 2.93748211e-30 1.00000000e+00 4.53999298e-05
  2.44260074e-36]
 [1.26641655e-14 5.10908903e-12 4.53999298e-05 1.00000000e+00
  4.24835426e-18]
 [8.62880116e-60 4.53999298e-05 2.44260074e-36 4.24835426e-18
  1.00000000e+00]]
[[ 1 -2  4]
 [ 1  4  1]
 [ 1  1  6]
 [ 1  2  4]
```

```
 [ 1  6  2]]
[[ 0. 45. 13. 16. 68.]
 [45.  0. 34. 13.  5.]
 [13. 34.  0.  5. 41.]
 [16. 13.  5.  0. 20.]
 [68.  5. 41. 20.  0.]]
[[1.00000000e+00 8.19401262e-40 5.10908903e-12 1.26641655e-14
  8.62880116e-60]
 [8.19401262e-40 1.00000000e+00 2.93748211e-30 5.10908903e-12
  4.53999298e-05]
 [5.10908903e-12 2.93748211e-30 1.00000000e+00 4.53999298e-05
  2.44260074e-36]
 [1.26641655e-14 5.10908903e-12 4.53999298e-05 1.00000000e+00
  4.24835426e-18]
 [8.62880116e-60 4.53999298e-05 2.44260074e-36 4.24835426e-18
  1.00000000e+00]]
[[ 1 -2  4]
 [ 1  4  1]
 [ 1  1  6]
 [ 1  2  4]
 [ 1  6  2]]
[[ 0. 45. 13. 16. 68.]
 [45.  0. 34. 13.  5.]
 [13. 34.  0.  5. 41.]
 [16. 13.  5.  0. 20.]
 [68.  5. 41. 20.  0.]]
[[1.00000000e+00 8.19401262e-40 5.10908903e-12 1.26641655e-14
  8.62880116e-60]
 [8.19401262e-40 1.00000000e+00 2.93748211e-30 5.10908903e-12
  4.53999298e-05]
 [5.10908903e-12 2.93748211e-30 1.00000000e+00 4.53999298e-05
  2.44260074e-36]
 [1.26641655e-14 5.10908903e-12 4.53999298e-05 1.00000000e+00
  4.24835426e-18]
 [8.62880116e-60 4.53999298e-05 2.44260074e-36 4.24835426e-18
  1.00000000e+00]]
Optimization terminated successfully

Alpha [1.19999999 1.19999999 0.79999999 0.79999999 0.79999999]
Weights [4.44089210e-16 4.79999996e+00 3.59999997e+00]
Bias -23.39999982565643

print("implementation.py indices of support vectors:", \
      np.array(range(y.shape[0]))[nonlin_svm.a>1e-8])
if (nonlin_ref.support_ != np.array(range(y.shape[0]))
[nonlin_svm.a>1e-8]).all():
    raise Exception("The calculation is wrong")

implementation.py indices of support vectors: [0 1 2 3 4]
```

```
# print("scikit-learn weights assigned to the features:",
nonlin_ref.coef_)
# print("implementation.py weights assigned to the features:",
nonlin_svm.w)

# diff = abs(nonlin_ref.coef_ - nonlin_svm.w)
# if (diff > 1e-3).any():
#     raise Exception("The calculation is wrong")
```

```
scikit-learn weights assigned to the features: [1. 1.]
implementation.py weights assigned to the features: [4.44089210e-16
4.79999996e+00 3.59999997e+00]
```

```
---------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
Cell In[58], line 4
      1 print("scikit-learn weights assigned to the features:",
nonlin_ref.class_weight_)
      2 print("implementation.py weights assigned to the features:",
nonlin_svm.w)
----> 4 diff = abs(nonlin_ref.class_weight_ - nonlin_svm.w)
      5 if (diff > 1e-3).any():
      6     raise Exception("The calculation is wrong")

ValueError: operands could not be broadcast together with shapes (2,)
(3,)
```

```
print("scikit-learn bias weight:", nonlin_ref.intercept_)
print("implementation.py bias weight:", nonlin_svm.b)

diff = abs(nonlin_ref.intercept_ - nonlin_svm.b)
if (diff > 1e-3).all():
    raise Exception("The calculation is wrong")
```

```
scikit-learn bias weight: [0.56782817]
implementation.py bias weight: -23.39999982565643
```

```
---------------------------------------------------------------------
-----
Exception                                 Traceback (most recent call
last)
Cell In[59], line 6
      4 diff = abs(nonlin_ref.intercept_ - nonlin_svm.b)
      5 if (diff > 1e-3).all():
----> 6     raise Exception("The calculation is wrong")

Exception: The calculation is wrong
```