1a) Implementing TISP variable selection for classification.

As always start with importing what we need and finding our data locations.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, roc_curve, auc

import os

os.chdir("C:/Users/rique/Downloads/gisette")
```

For part a) We're using the Gisette data

load the data.

```
g_train_data = np.loadtxt("gisette_train.data")
g_train_labels = np.loadtxt("gisette_train.labels")
g_valid_data = np.loadtxt("gisette_valid.data")
g_valid_labels = np.loadtxt("gisette_valid.labels")
```

Normalize the features. Copying and pasting from my HW3, cause it works. Don't fix what ain't broken.

```
std=np.std(g_train_data, axis=0)

#set a mask so we dont get a divide by standard deviation of zero
mask = (std != 0)

#apply the mask, get the mean and standard dev of the normalized data
g_train_data = g_train_data[:, mask]
mean = np.mean(g_train_data, axis=0)
true_std = np.std(g_train_data, axis=0)

#standardize the features in both training and test datasets
g_train_data = (g_train_data-mean)/true_std
g_valid_data = g_valid_data[:, mask]
g_valid_data = (g_valid_data-mean)/true_std

#add a bias term
g_train_data = np.insert(g_train_data, 0, 1, axis=1)
g_valid_data = np.insert(g_valid_data, 0, 1, axis=1)
```

```
g_train_labels[g_train_labels == 0] = -1
g_valid_labels[g_valid_labels == 0] = -1
```

Verifying data has mean 0 and variance of 1

```
print("Train mean: ", np.mean(g_train_data))
print("Train variance: ", np.var(g_train_data))
print("Test mean: ", np.mean(g_valid_data))
print("Test variance: ", np.var(g_valid_data))
```

```
Train mean:  0.0002017756255044388
Train variance:  0.9999999592865941
Test mean:  0.0062998654925201245
Test variance:  1.0634350583552006
```

Initalize our needed parameters

threshold values are subject to change.... ALOT.

```
iterations = 100

#lambda; these values are subject to change to find out features


#For lambda of: 0.087959 feature is: 98
#0.08795105 101



#For lambda of: 0.038549 feature is: 499
#For lambda of: 0.038535 feature is: 502
#For lambda of: 0.038541 feature is: 502



thresholds = [0.19, 0.133, 0.08795105, 0.05291, 0.038545]

w = np.zeros(g_train_data.shape[1])

train_misclass_errors = []
valid_misclass_errors = []
train_misclass_errors_30 = []
features = []
fpr_train_list = []
tpr_train_list = []
roc_auc_train_list = []
fpr_valid_list = []
tpr_valid_list = []
roc_auc_valid_list = []
```

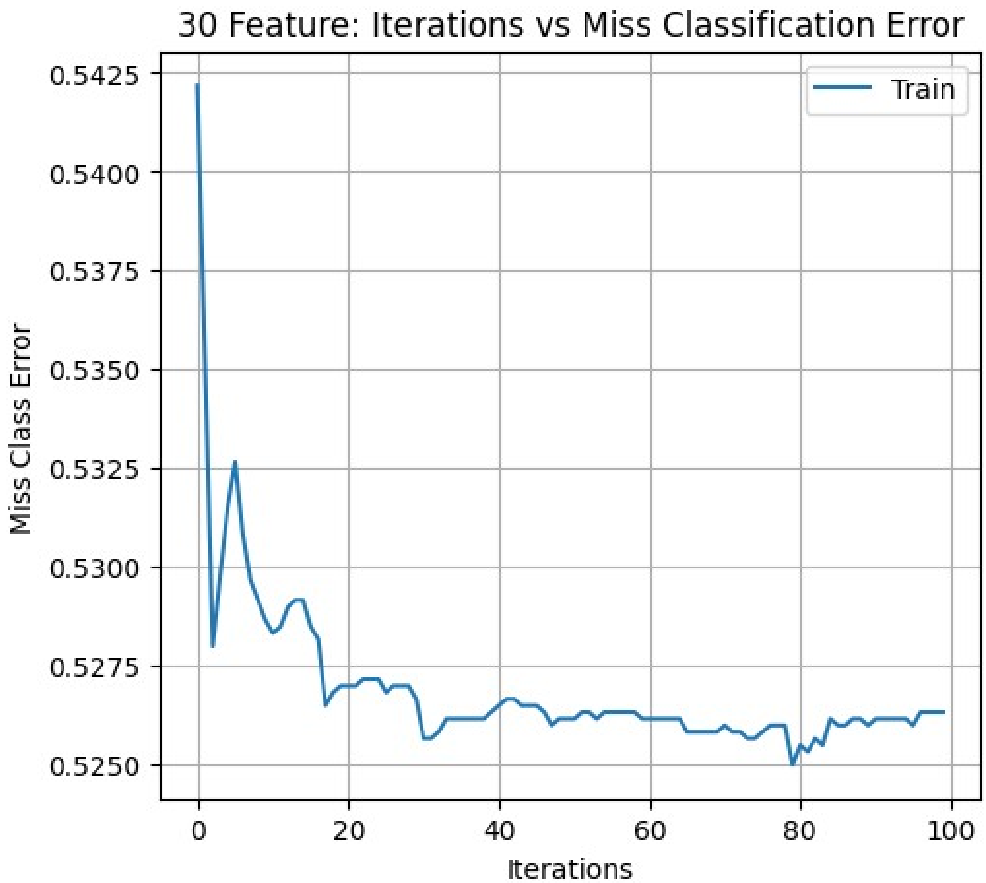
TISP implementation

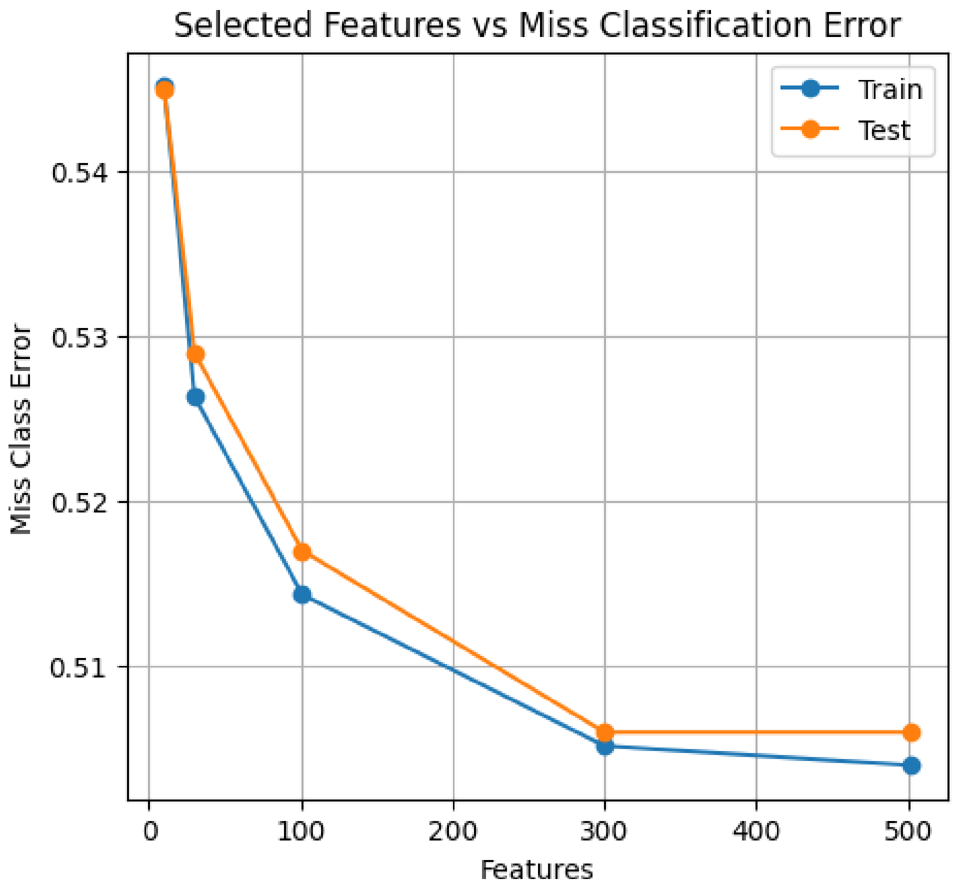
```
for lambda_ in thresholds:
    for i in range(iterations):
        # Dot product of train data and weight
        dot = np.sum(g_train_data * w, axis=1)

        # Gradient
        gradient = np.sum((g_train_labels / (1 + np.exp(g_train_labels
* dot))) * (g_train_data).T, axis=1)

        # Update the weight with our gradient
        w += gradient * (1 / g_train_data.shape[0])
        w[np.absolute(w) <= lambda_] = 0

      #  print(i, "weight is: ", np.sum(w != 0))

        # Recalculate dot product of train data and updated weight
        dot = np.sum(g_train_data * w, axis=1)

        # Prediction here is based on if the dot product of train/test
sets is greater than zero
        y_pred_train = ((dot >= 0) == g_train_labels)
        misclass_error_train = 1 - accuracy_score(g_train_labels,
y_pred_train)

         #cause we wanna plot for 30 features
        if(lambda_ == 0.133):
            train_misclass_errors_30.append(misclass_error_train)

    feature = np.sum(w != 0)
    features.append(feature)
   # print("For lambda of:", lambda_, "feature is:", feature)

    train_misclass_errors.append(misclass_error_train)

    dot_valid = np.sum(g_valid_data * w, axis=1)
    y_pred_valid = ((dot_valid >= 0) == g_valid_labels)
    misclass_error_valid = 1 - accuracy_score(g_valid_labels,
y_pred_valid)
    valid_misclass_errors.append(misclass_error_valid)

    # Calculate ROC curve values for the training set
    fpr_train, tpr_train, _ = roc_curve(g_train_labels, 1 / (1 +
    np.exp(-dot)))
    roc_auc_train = auc(fpr_train, tpr_train)
    fpr_train_list.append(fpr_train)
    tpr_train_list.append(tpr_train)
    roc_auc_train_list.append(roc_auc_train)

    # Calculate ROC curve values for the validation set
```

```
    fpr_valid, tpr_valid, _ = roc_curve(g_valid_labels, 1 / (1 +
    np.exp(-dot_valid)))
    roc_auc_valid = auc(fpr_valid, tpr_valid)
    fpr_valid_list.append(fpr_valid)
    tpr_valid_list.append(tpr_valid)
    roc_auc_valid_list.append(roc_auc_valid)

    w=np.zeros_like(w)

#print("Features selected:", features)
#print("Train misclassification errors:", train_misclass_errors)
#print("Validation misclassification errors:", valid_misclass_errors)
```

Plot the stuff

```
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(range(iterations), train_misclass_errors_30, label="Train")
plt.xlabel('Iterations')
plt.ylabel('Miss Class Error')
plt.title('30 Feature: Iterations vs Miss Classification Error')
plt.grid()
plt.legend()

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(features, train_misclass_errors, marker="o", label="Train")
plt.plot(features, valid_misclass_errors, marker="o", label="Test")
plt.xlabel('Features')
plt.ylabel('Miss Class Error')
plt.title('Selected Features vs Miss Classification Error')
plt.grid()
plt.legend()

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 2)
plt.plot(fpr_train_list[-1], tpr_train_list[-1], color='blue', lw=2,
label=f'Training Set (AUC = {roc_auc_train_list[-1]:.2f})')
plt.plot(fpr_valid_list[-1], tpr_valid_list[-1], color='darkorange',
lw=2, label=f'Validation Set (AUC = {roc_auc_valid_list[-1]:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='No
Discrimination')

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()


plt.show()
```


30 Feature: Iterations vs Miss Classification Error
Train
Miss Class Error
0.5425
0.5400
0.5375
0.5350
0.5325
0.5300
0.5275
0.5250
0
20
40
60
80
100
Iterations


Selected Features vs Miss Classification Error
Train
Test
Miss Class Error
0.54
0.53
0.52
0.51
0
100
200
300
400
500
Features

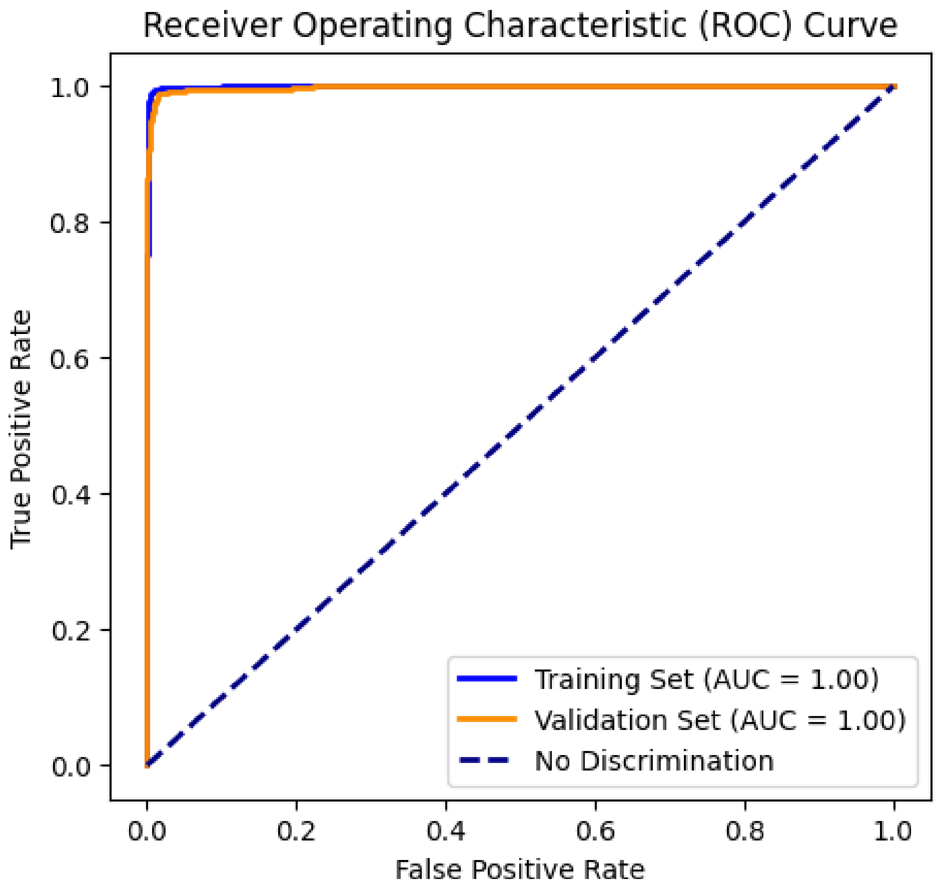


Table with the Misclass errors, lambda, features on test and training sets

```
results = pd.DataFrame({
    'Lambda': thresholds,
    'Features': features,
    'Train MisClass Error': train_misclass_errors,
    'Test MisClass Error': valid_misclass_errors
})

print(results)
```

```
     Lambda  Features  Train MisClass Error  Test MisClass Error
0  0.190000        10              0.545167                0.545
1  0.133000        30              0.526333                0.529
2  0.087951       101              0.514333                0.517
3  0.052910       300              0.505167                0.506
4  0.038545       501              0.504000                0.506
```

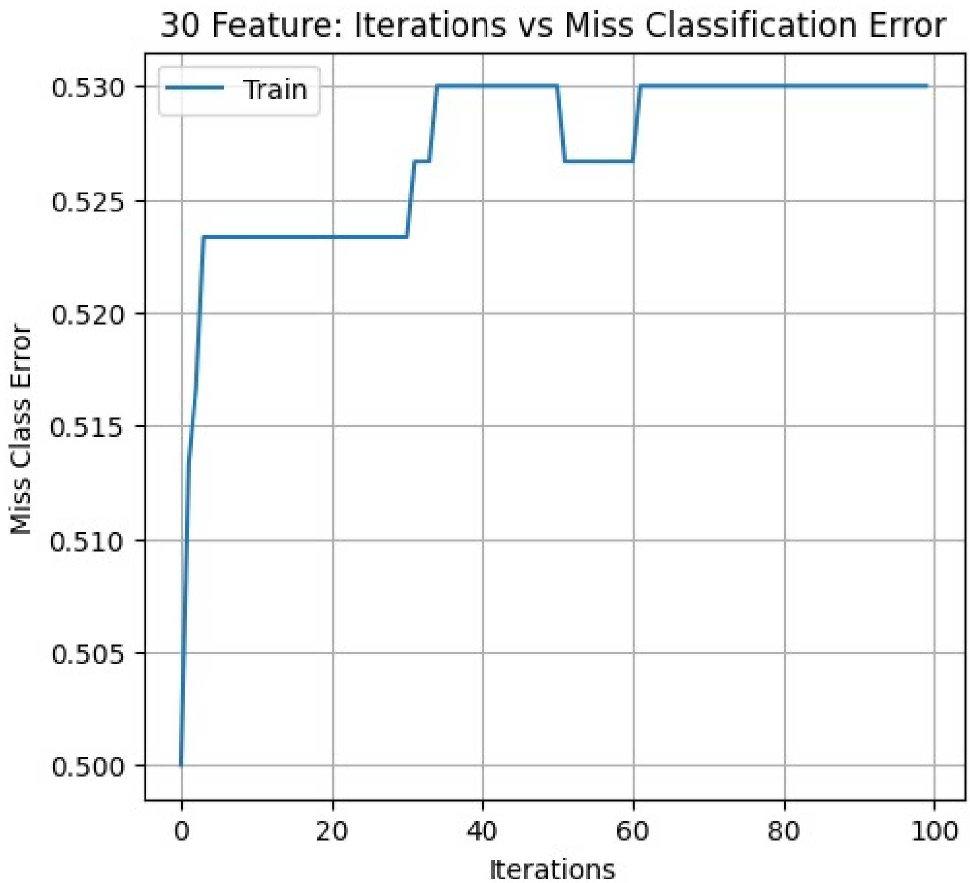
1b) Repeating a) but on the dexter set.

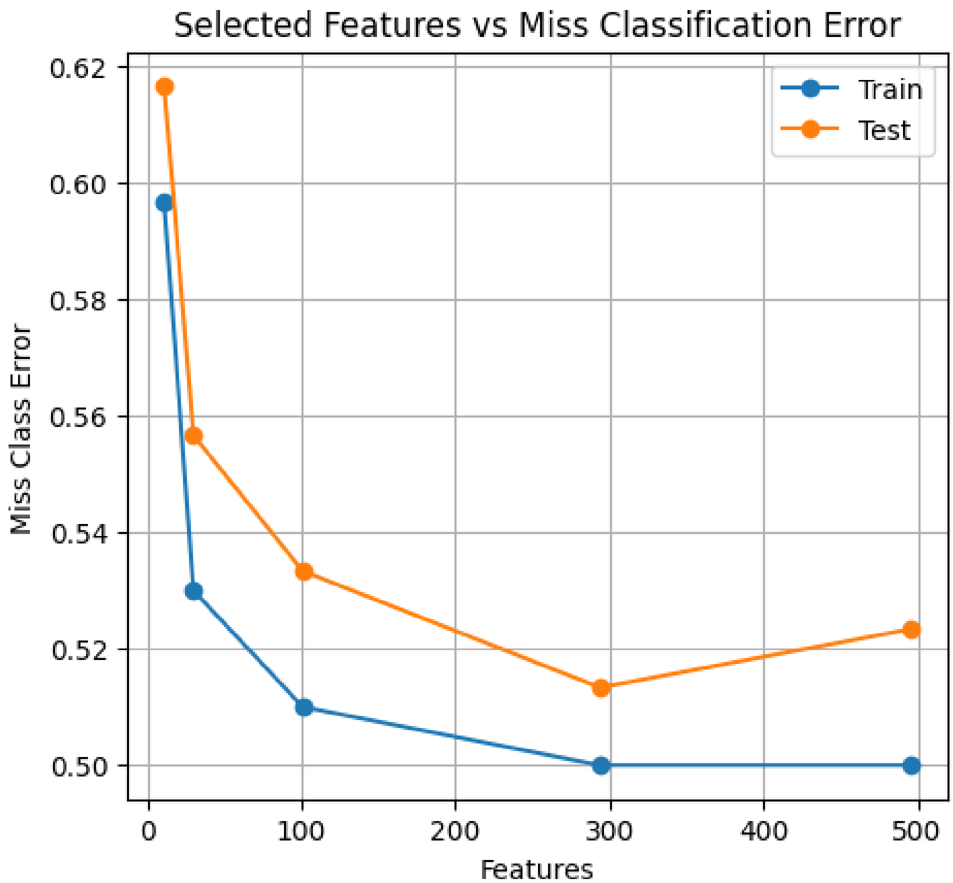
Copy & Pasting above code, with only differences being the dataset, and another guessing game with the thresholds

```
os.chdir("C:/Users/rique/Downloads/dexter")

X = np.genfromtxt('dexter_train.csv', delimiter=',')
y = np.loadtxt('dexter_train.labels')

Xtest = np.genfromtxt('dexter_valid.csv', delimiter=',')
yTest = np.loadtxt('dexter_valid.labels')

std=np.std(X, axis=0)

#set a mask so we dont get a divide by standard deviation of zero
mask = (std != 0)

#apply the mask, get the mean and standard dev of the normalized data
X = X[:, mask]
mean = np.mean(X, axis=0)
true_std = np.std(X, axis=0)

#standardize the features in both training and test datasets
X = (X-mean)/true_std
Xtest = Xtest[:, mask]
Xtest = (Xtest-mean)/true_std

#add a bias term
X = np.insert(X, 0, 1, axis=1)
Xtest = np.insert(Xtest, 0, 1, axis=1)

y[y == 0] = -1
yTest[yTest == 0] = -1

iterations = 100

#lambda; these values are subject to change to find out features

thresholds = [0.14, 0.098, 0.071, 0.0525, 0.0468]

w = np.zeros(X.shape[1])

train_misclass_errors = []
valid_misclass_errors = []
train_misclass_errors_30 = []
features = []
fpr_train_list = []
tpr_train_list = []
roc_auc_train_list = []
```

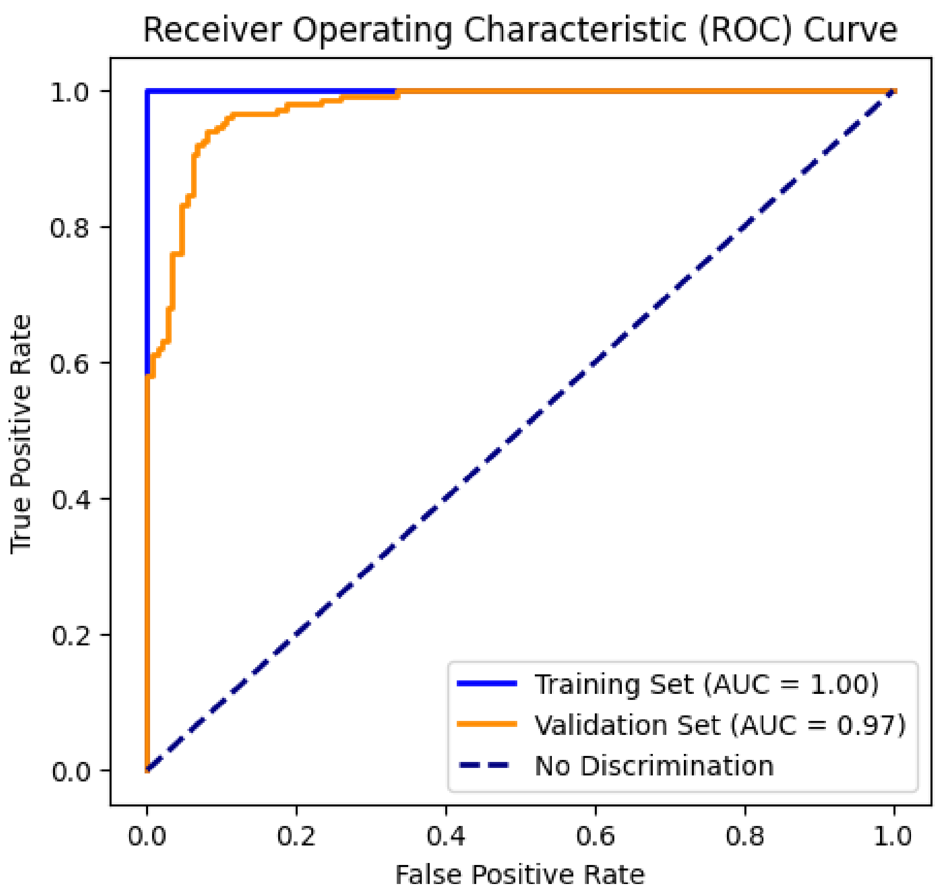
```
fpr_valid_list = []
tpr_valid_list = []
roc_auc_valid_list = []

for lambda_ in thresholds:
    for i in range(iterations):
        # Dot product of train data and weight
        dot = np.sum(X * w, axis=1)

        # Gradient
        gradient = np.sum((y / (1 + np.exp(y * dot))) * (X).T, axis=1)

        # Update the weight with our gradient
        w += gradient * (1 / X.shape[0])
        w[np.absolute(w) <= lambda_] = 0

        #print(i, "weight is: ", np.sum(w != 0))

        # Recalculate dot product of train data and updated weight
        dot = np.sum(X * w, axis=1)

        # Prediction here is based on if the dot product of train/test
sets is greater than zero
        y_pred_train = ((dot >= 0) == y)
        misclass_error_train = 1 - accuracy_score(y, y_pred_train)

         #cause we wanna plot for 30 features
        if(lambda_ == 0.098):
            train_misclass_errors_30.append(misclass_error_train)

    feature = np.sum(w != 0)
    features.append(feature)
  #  print("For lambda of:", lambda_, "feature is:", feature)

    train_misclass_errors.append(misclass_error_train)

    dot_valid = np.sum(Xtest * w, axis=1)
    y_pred_valid = ((dot_valid >= 0) == yTest)
    misclass_error_valid = 1 - accuracy_score(yTest, y_pred_valid)
    valid_misclass_errors.append(misclass_error_valid)

    # Calculate ROC curve values for the training set
    fpr_train, tpr_train, _ = roc_curve(y, 1 / (1 +
    np.exp(-dot)))
    roc_auc_train = auc(fpr_train, tpr_train)
    fpr_train_list.append(fpr_train)
    tpr_train_list.append(tpr_train)
    roc_auc_train_list.append(roc_auc_train)

    # Calculate ROC curve values for the validation set
```

```
    fpr_valid, tpr_valid, _ = roc_curve(yTest, 1 / (1 +
    np.exp(-dot_valid)))
    roc_auc_valid = auc(fpr_valid, tpr_valid)
    fpr_valid_list.append(fpr_valid)
    tpr_valid_list.append(tpr_valid)
    roc_auc_valid_list.append(roc_auc_valid)

    w=np.zeros_like(w)

#print("Features selected:", features)
#print("Train misclassification errors:", train_misclass_errors)
#print("Validation misclassification errors:", valid_misclass_errors)

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(range(iterations), train_misclass_errors_30, label="Train")
plt.xlabel('Iterations')
plt.ylabel('Miss Class Error')
plt.title('30 Feature: Iterations vs Miss Classification Error')
plt.grid()
plt.legend()

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(features, train_misclass_errors, marker="o", label="Train")
plt.plot(features, valid_misclass_errors, marker="o", label="Test")
plt.xlabel('Features')
plt.ylabel('Miss Class Error')
plt.title('Selected Features vs Miss Classification Error')
plt.grid()
plt.legend()

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 2)
plt.plot(fpr_train_list[-1], tpr_train_list[-1], color='blue', lw=2,
label=f'Training Set (AUC = {roc_auc_train_list[-1]:.2f})')
plt.plot(fpr_valid_list[-1], tpr_valid_list[-1], color='darkorange',
lw=2, label=f'Validation Set (AUC = {roc_auc_valid_list[-1]:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='No
Discrimination')

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()


plt.show()
```


30 Feature: Iterations vs Miss Classification Error
Train
Miss Class Error
0.530
0.525
0.520
0.515
0.510
0.505
0.500
0
20
40
60
80
100
Iterations



Selected Features vs Miss Classification Error
Train
Test
Miss Class Error
0.62
0.60
0.58
0.56
0.54
0.52
0.50
0
100
200
300
400
500
Features




```python
results = pd.DataFrame({
    'Lambda': thresholds,
    'Features': features,
    'Train MisClass Error': train_misclass_errors,
    'Test MisClass Error': valid_misclass_errors
})

print(results)
```

	Lambda	Features	Train MisClass Error	Test MisClass Error
0	0.1400	11	0.596667	0.616667
1	0.0980	30	0.530000	0.556667
2	0.0710	101	0.510000	0.533333
3	0.0525	294	0.500000	0.513333
4	0.0468	495	0.500000	0.523333

1c) Repeating a) and b) but on the madelon set.

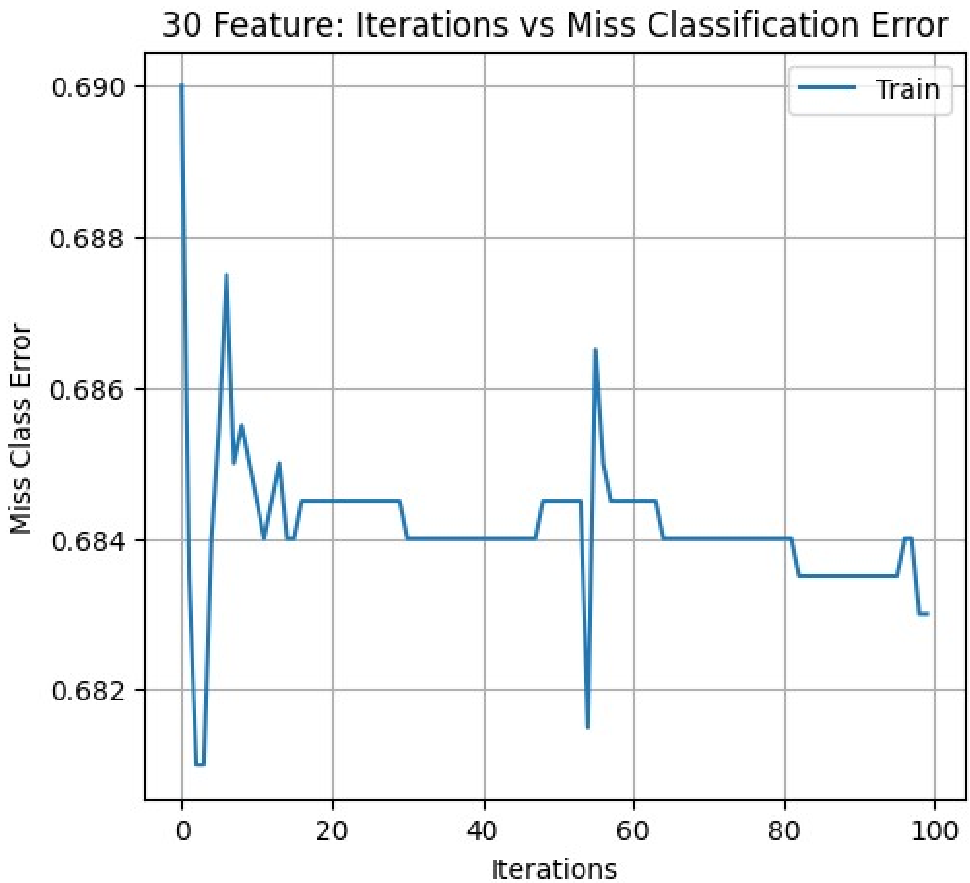
Copy & Pasting above code, with only differences being the dataset, and yet another tedious guessing game with the thresholds

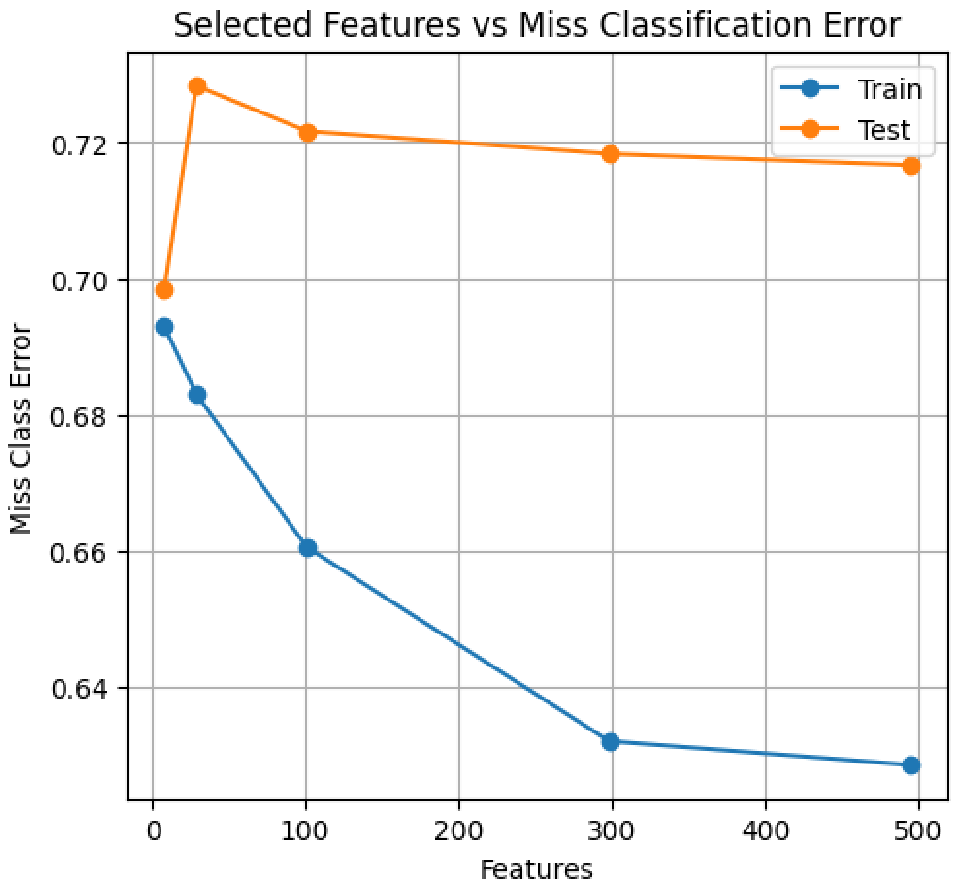
```
os.chdir("C:/Users/rique/Downloads/MADELON")

X = np.loadtxt("madelon_train.data")
y = np.loadtxt("madelon_train.labels")

Xtest = np.loadtxt("madelon_valid.data")
yTest = np.loadtxt("madelon_valid.labels")

std=np.std(X, axis=0)

#set a mask so we dont get a divide by standard deviation of zero
mask = (std != 0)

#apply the mask, get the mean and standard dev of the normalized data
X = X[:, mask]
mean = np.mean(X, axis=0)
true_std = np.std(X, axis=0)

#standardize the features in both training and test datasets
X = (X-mean)/true_std
Xtest = Xtest[:, mask]
Xtest = (Xtest-mean)/true_std

#add a bias term
X = np.insert(X, 0, 1, axis=1)
Xtest = np.insert(Xtest, 0, 1, axis=1)

y[y == 0] = -1
yTest[yTest == 0] = -1

iterations = 100

#lambda; these values are subject to change to find out features

#For lambda of: 0.0009 feature is: 473


thresholds = [0.02999, 0.0245, 0.017, 0.0075, 0.000199]

w = np.zeros(X.shape[1])

train_misclass_errors = []
valid_misclass_errors = []
train_misclass_errors_30 = []
features = []
```

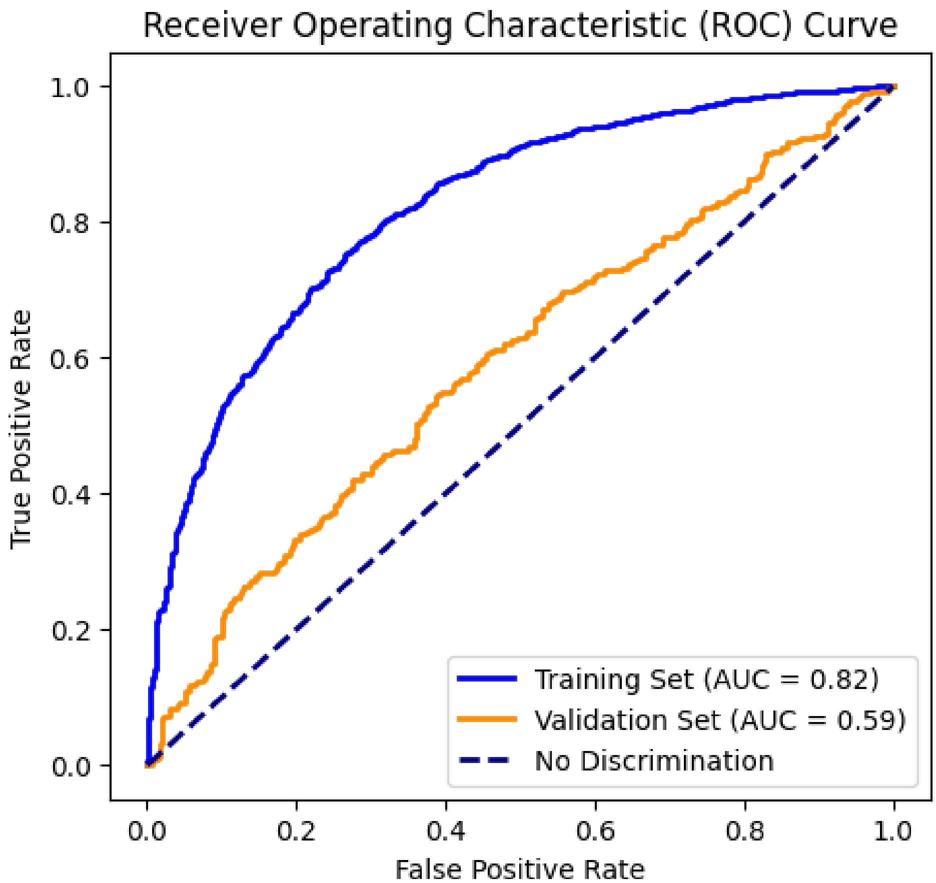
```python
fpr_train_list = []
tpr_train_list = []
roc_auc_train_list = []
fpr_valid_list = []
tpr_valid_list = []
roc_auc_valid_list = []

for lambda_ in thresholds:
    for i in range(iterations):
        # Dot product of train data and weight
        dot = np.sum(X * w, axis=1)

        # Gradient
        gradient = np.sum((y / (1 + np.exp(y * dot))) * (X).T, axis=1)

        # Update the weight with our gradient
        w += gradient * (1 / X.shape[0])
        w[np.absolute(w) <= lambda_] = 0

       # print(i, "weight is: ", np.sum(w != 0))

        # Recalculate dot product of train data and updated weight
        dot = np.sum(X * w, axis=1)

        # Prediction here is based on if the dot product of train/test
sets is greater than zero
        y_pred_train = ((dot >= 0) == y)
        misclass_error_train = 1 - accuracy_score(y, y_pred_train)

         #cause we wanna plot for 30 features
        if(lambda_ == 0.0245):
            train_misclass_errors_30.append(misclass_error_train)

    feature = np.sum(w != 0)
    features.append(feature)
    #print("For lambda of:", lambda_, "feature is:", feature)

    train_misclass_errors.append(misclass_error_train)

    dot_valid = np.sum(Xtest * w, axis=1)
    y_pred_valid = ((dot_valid >= 0) == yTest)
    misclass_error_valid = 1 - accuracy_score(yTest, y_pred_valid)
    valid_misclass_errors.append(misclass_error_valid)

    # Calculate ROC curve values for the training set
    fpr_train, tpr_train, _ = roc_curve(y, 1 / (1 +
    np.exp(-dot)))
    roc_auc_train = auc(fpr_train, tpr_train)
    fpr_train_list.append(fpr_train)
    tpr_train_list.append(tpr_train)
```

```
    roc_auc_train_list.append(roc_auc_train)

    # Calculate ROC curve values for the validation set
    fpr_valid, tpr_valid, _ = roc_curve(yTest, 1 / (1 +
    np.exp(-dot_valid)))
    roc_auc_valid = auc(fpr_valid, tpr_valid)
    fpr_valid_list.append(fpr_valid)
    tpr_valid_list.append(tpr_valid)
    roc_auc_valid_list.append(roc_auc_valid)

    w=np.zeros_like(w)

#print("Features selected:", features)
#print("Train misclassification errors:", train_misclass_errors)
#print("Validation misclassification errors:", valid_misclass_errors)

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(range(iterations), train_misclass_errors_30, label="Train")
plt.xlabel('Iterations')
plt.ylabel('Miss Class Error')
plt.title('30 Feature: Iterations vs Miss Classification Error')
plt.grid()
plt.legend()

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(features, train_misclass_errors, marker="o", label="Train")
plt.plot(features, valid_misclass_errors, marker="o", label="Test")
plt.xlabel('Features')
plt.ylabel('Miss Class Error')
plt.title('Selected Features vs Miss Classification Error')
plt.grid()
plt.legend()

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 2)
plt.plot(fpr_train_list[-1], tpr_train_list[-1], color='blue', lw=2,
label=f'Training Set (AUC = {roc_auc_train_list[-1]:.2f})')
plt.plot(fpr_valid_list[-1], tpr_valid_list[-1], color='darkorange',
lw=2, label=f'Validation Set (AUC = {roc_auc_valid_list[-1]:.2f})')
# Add a dashed diagonal line (no-discrimination ROC curve)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='No
Discrimination')

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
```

```
plt.show()
```


30 Feature: Iterations vs Miss Classification Error
Train
0.690
0.688
0.686
0.684
0.682
Miss Class Error
0
20
40
60
80
100
Iterations


Selected Features vs Miss Classification Error
Train
Test
0.72
0.70
0.68
0.66
0.64
Miss Class Error
0
100
200
300
400
500
Features




```
results = pd.DataFrame({
    'Lambda': thresholds,
    'Features': features,
    'Train MisClass Error': train_misclass_errors,
    'Test MisClass Error': valid_misclass_errors
})

print(results)
```

```
     Lambda  Features  Train MisClass Error  Test MisClass Error
0  0.029990         8                0.6930             0.698333
1  0.024500        29                0.6830             0.728333
2  0.017000       101                0.6605             0.721667
3  0.007500       298                0.6320             0.718333
4  0.000199       494                0.6285             0.716667
```