1a) Implementing FSA variable selection for linear models & binary 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/datasets")
```

For part a) We're using the Gisette data

load the data.

```
def load_dataset(X, y, Xtest, yTest, probChar):

    if(probChar == 'a'):
        X = np.loadtxt("gisette_train.data")
        y = np.loadtxt("gisette_train.labels")
        Xtest = np.loadtxt("gisette_valid.data")
        yTest = np.loadtxt("gisette_valid.labels")
    if(probChar == 'b'):
        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')
    if(probChar == 'c'):
        X = np.loadtxt("madelon_train.data")
        y = np.loadtxt("madelon_train.labels")
        Xtest = np.loadtxt("madelon_valid.data")
        yTest = np.loadtxt("madelon_valid.labels")

    return X, y, Xtest, yTest
```

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

```
def normalize(X, y, Xtest, yTest):

    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

    return X, y, Xtest, yTest
```

Logistic Loss from Slide 4 of the FSA lecture notes

1/N * sigma * ln(1 + e^-yw^Tx) + s||w||_2^2

```
def log_loss(X, y, w, s):
    #predict
    dot = X@w


    #logistic loss
    loss = np.mean(np.log(1 + np.exp(-y * dot.squeeze())))


    #penalty
    L2_ridge = s * np.sum(np.abs(w)**2)


    return loss + L2_ridge
```

Fit to Compute Pred

from my HW2

```
def fit(X, y, lambda_):
    X_transpose = X.T
```

```
    XTX = np.dot(X_transpose, X)
    matrix = lambda_ * np.identity(XTX.shape[0])
    XTX_lambda = XTX + matrix
    XTy = np.dot(X_transpose, y)
    coefficients = np.linalg.solve(XTX_lambda, XTy)

    return coefficients
```

FSA using the variable eliminator and sorting

```
def FSA(X, y, Xtest, yTest, w, s, mu, lamb, k_features, iterations):

    for i in range(len(k_features)):
        for j in range(iterations):

            #############  COPIED FROM MY HW 3, CAUSE NOTHING ELSE WAS
WORKING
            z = np.dot(X, w)
            predictions = 1 / (1 + np.exp(-z))
            # Calculate the gradient of the logistic loss with L1
regularization
            gradient = np.dot(X.T, (predictions - y))/ len(y)
            l1_penalty = lamb * np.sign(w)
            gradient += l1_penalty
            # Update weights using gradient descent
            w -= lamb * gradient

            # Variable Schedule Eliminator from FSA lecture notes.
            el=int(k_features[i]+(p-k_features[i])*max(0.,(iterations-
2*j)/(2*j*mu+iterations)))

            print(i, 'el', el)
            wSort=np.argsort(np.absolute(w))
            wSort=wSort[-el:]
            w=w[wSort]
            X=(X.T[wSort]).T
            Xtest=(Xtest.T[wSort]).T

            #######FOR LOGISTIC LOSS
            if(k_features[i] == 30):
                loss = log_loss(X, y, w, s)
                #print("loss:", loss)
                train_loss_30.append(loss)

            ######FOR MISSCLASSIFICATION

        # Prediction here is based on if the dot product of train/test
sets is greater than zero
        y_pred_train = np.dot(X, fit(X, y, lamb))
        y_pred_binary = [1 if pred >= lamb else 0 for pred in
```

```
y_pred_train]

        misclass_error_train = 1 - accuracy_score(y, y_pred_binary)

        y_pred_valid = np.dot(Xtest, fit(Xtest, yTest, lamb))
        y_pred_valid_binary = [1 if pred >= lamb else 0 for pred in
y_pred_valid]

        misclass_error_valid = 1 -
accuracy_score(yTest,y_pred_valid_binary)

        train_misclass_errors.append(misclass_error_train)
        valid_misclass_errors.append(misclass_error_valid)

        dot = np.sum(X * w, axis=1)
        dot_valid = np.sum(Xtest * w, axis=1)

        if(k_features[i] == 100):
            # 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)

X = np.empty(1)
y = np.empty(1)
Xtest = np.empty(1)
yTest = np.empty(1)
probChar = 'a'

X, y, Xtest, yTest = load_dataset(X, y, Xtest, yTest, probChar)

X, y, Xtest, yTest = normalize(X, y, Xtest, yTest)
```

Verifying data has mean 0 and variance of 1

```
print("Train mean: ", np.mean(X))
print("Train variance: ", np.var(X))
print("Test mean: ", np.mean(Xtest))
print("Test variance: ", np.var(Xtest))
```

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

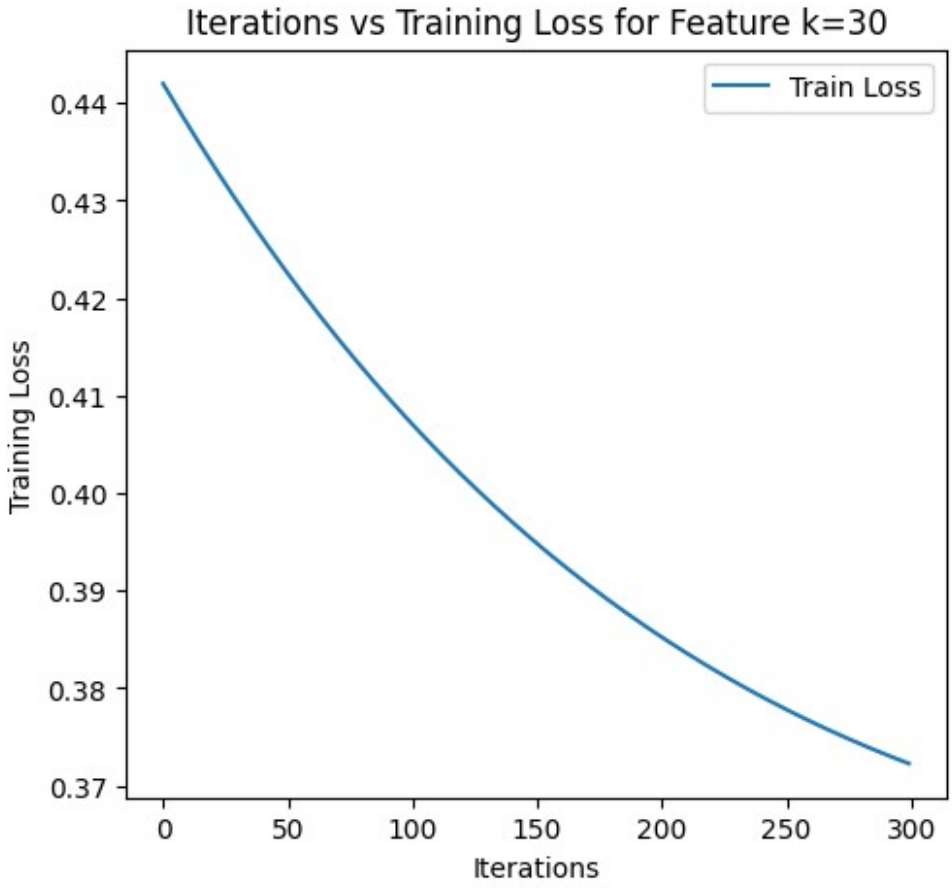
Initalize our needed parameters

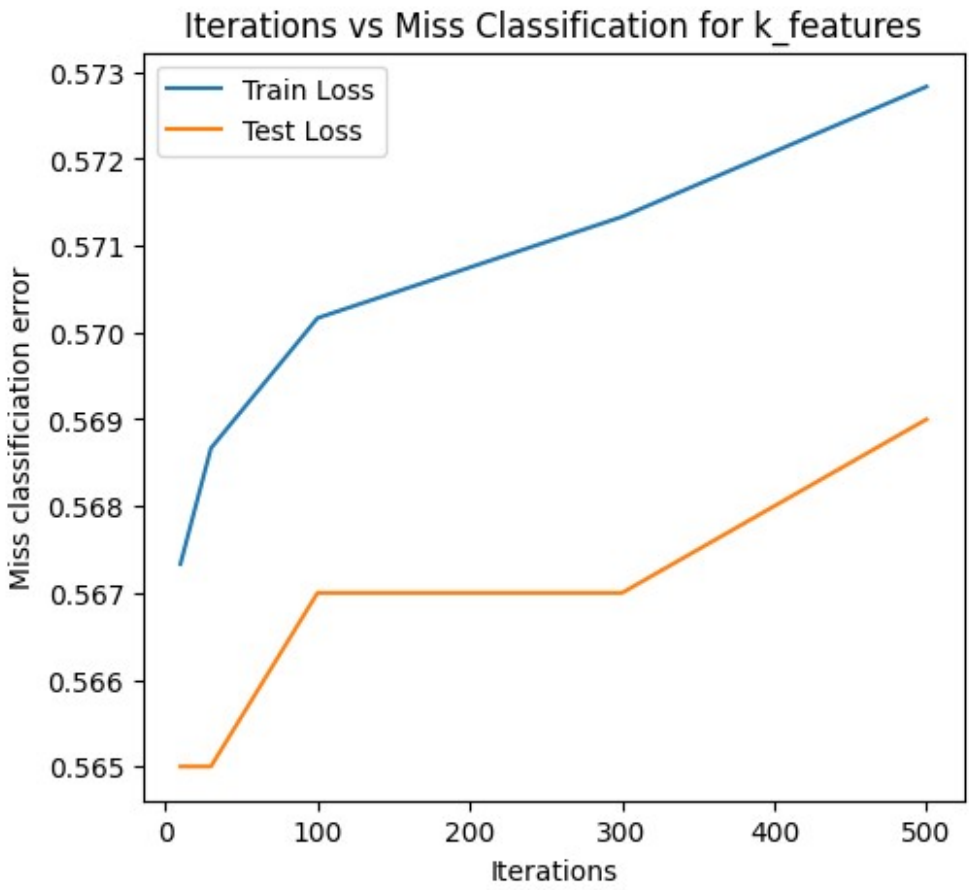
```
iterations = 300
s = 0.0001
mu = 200

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

k_features = [10,30,100,300,500]
lamb = 0.001

train_misclass_errors = []
valid_misclass_errors = []
train_loss_30 = []
fpr_train_list = []
tpr_train_list = []
roc_auc_train_list = []
fpr_valid_list = []
tpr_valid_list = []
roc_auc_valid_list = []

FSA(X, y, Xtest, yTest, w, s, mu, lamb, k_features, iterations)
```

Plot the stuff

```
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(range(iterations), train_loss_30, label="Train Loss")
plt.xlabel('Iterations')
plt.ylabel('Training Loss')
plt.title('Iterations vs Training Loss for Feature k=30')
plt.legend()

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(k_features, train_misclass_errors, label="Train Loss")
plt.plot(k_features, valid_misclass_errors, label="Test Loss")
plt.xlabel('Iterations')
plt.ylabel('Miss classificiation error')
plt.title('Iterations vs Miss Classification for k_features')
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.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()


plt.show()
```




Iterations vs Miss Classification for k_features
Train Loss
Test Loss
Miss classificiation error
Iterations
0.573
0.572
0.571
0.570
0.569
0.568
0.567
0.566
0.565
0
100
200
300
400
500

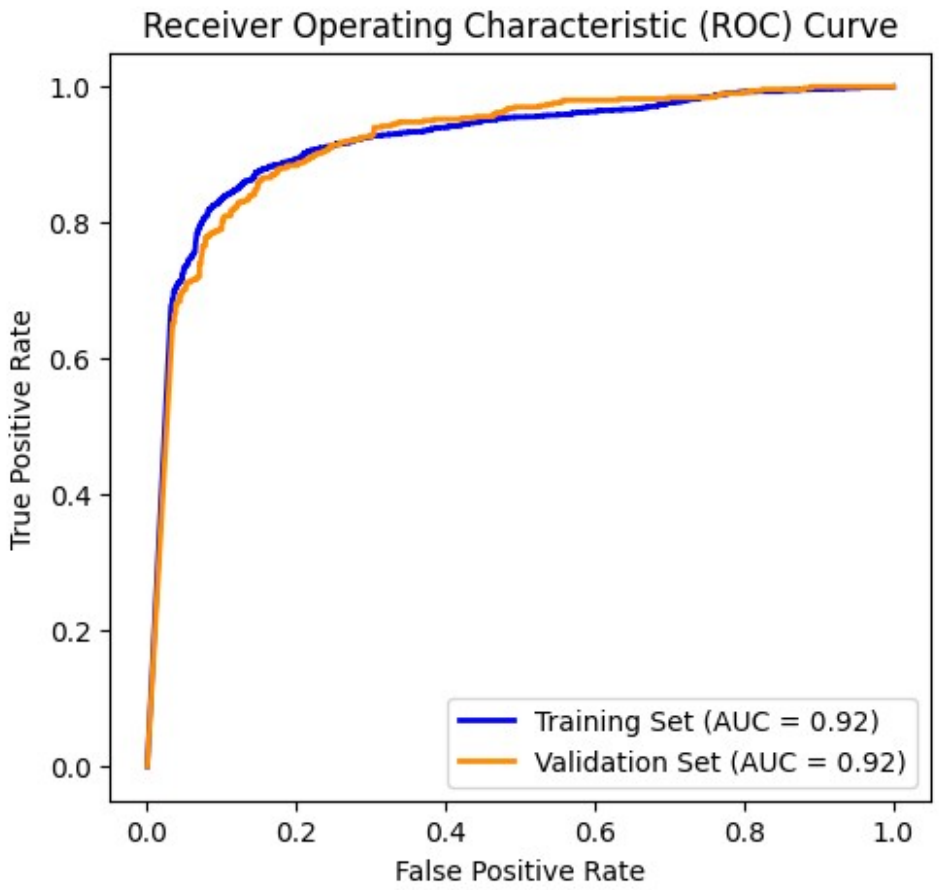


Table with the Misclass errors, features on test and training sets

```
results = pd.DataFrame({
    'K-features:': k_features,
    'Train MisClass Error:': train_misclass_errors,
    'Test MisClass Error:': valid_misclass_errors
})

print(results)
```

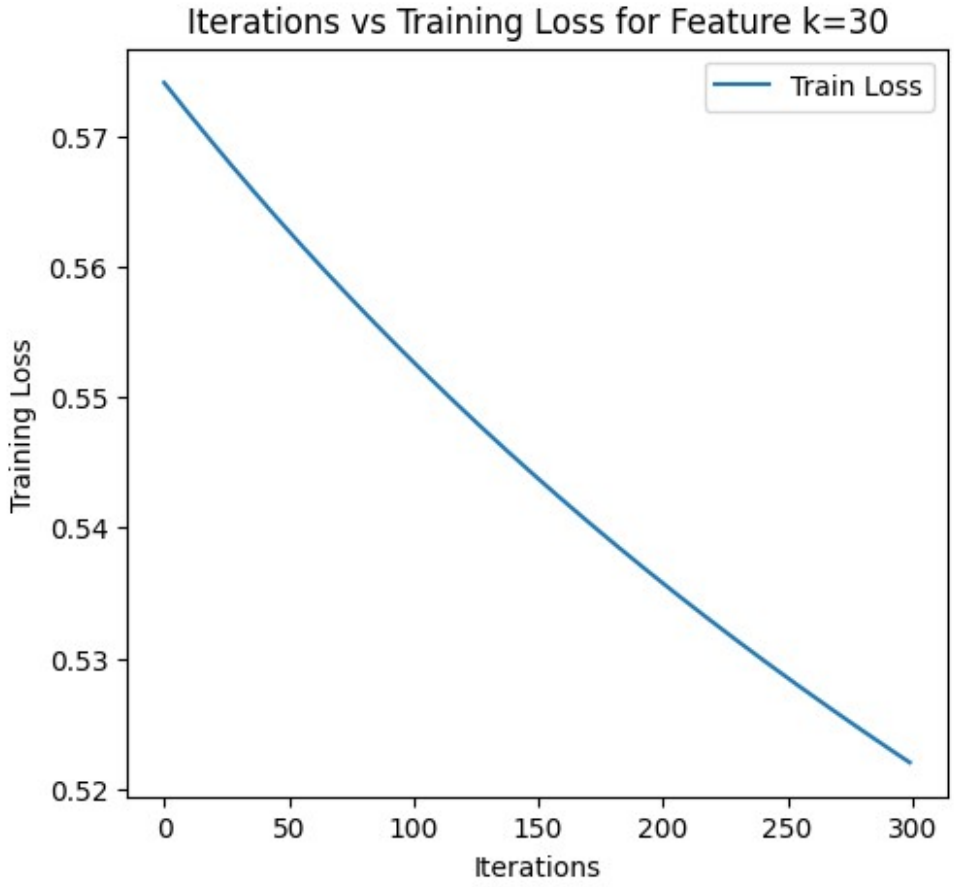
```
   K-features:  Train MisClass Error:  Test MisClass Error:
0           10                0.567333                 0.565
1           30                0.568667                 0.565
2          100                0.570167                 0.567
3          300                0.571333                 0.567
4          500                0.572833                 0.569
```

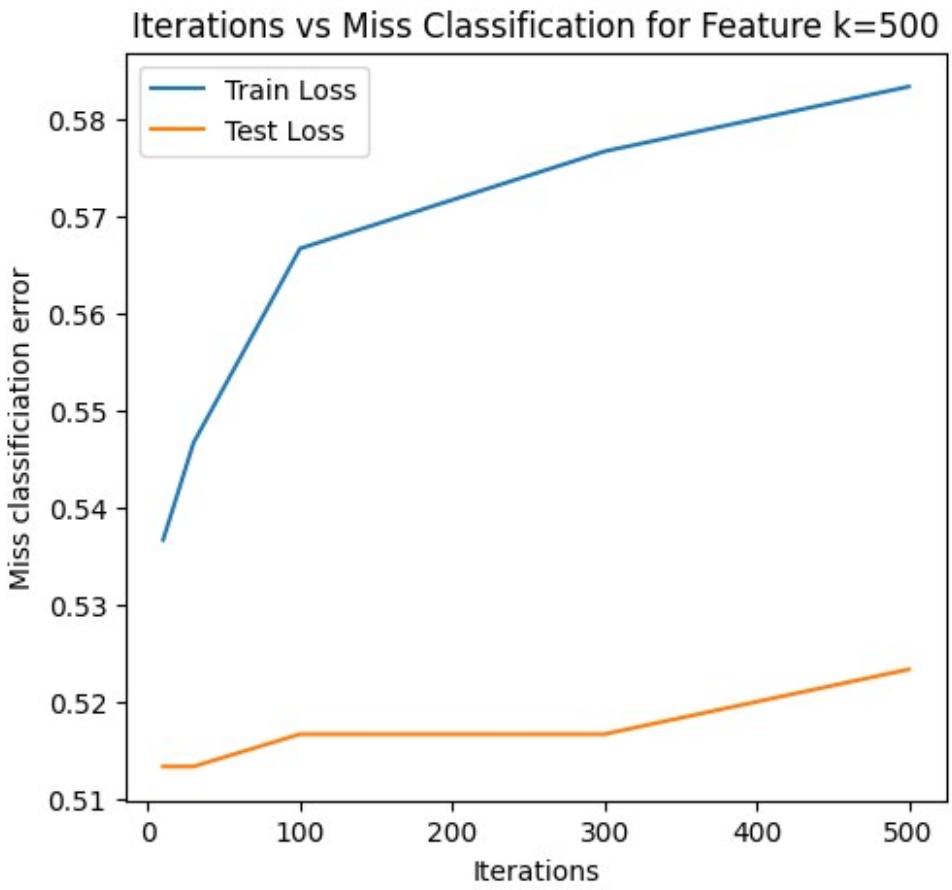
1b) DEXTER DATASET

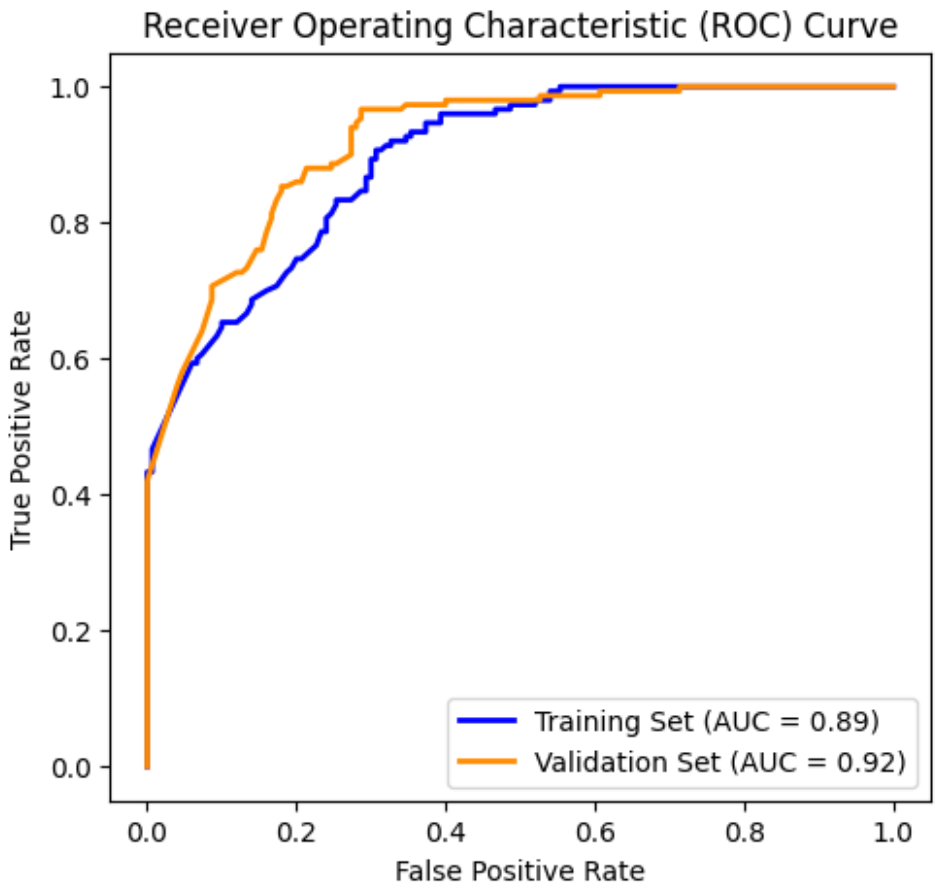
```
X = np.empty(1)
y = np.empty(1)
```

```
Xtest = np.empty(1)
yTest = np.empty(1)
probChar = 'b'

X, y, Xtest, yTest = load_dataset(X, y, Xtest, yTest, probChar)
X, y, Xtest, yTest = normalize(X, y, Xtest, yTest)

iterations = 300
s = 0.0001
mu = 200

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

k_features = [10,30,100,300,500]
lamb = 0.001

train_misclass_errors = []
valid_misclass_errors = []
train_loss_30 = []
fpr_train_list = []
tpr_train_list = []
roc_auc_train_list = []
fpr_valid_list = []
tpr_valid_list = []
roc_auc_valid_list = []

FSA(X, y, Xtest, yTest, w, s, mu, lamb, k_features, iterations)

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(range(iterations), train_loss_30, label="Train Loss")
plt.xlabel('Iterations')
plt.ylabel('Training Loss')
plt.title('Iterations vs Training Loss for Feature k=30')
plt.legend()

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(k_features, train_misclass_errors, label="Train Loss")
plt.plot(k_features, valid_misclass_errors, label="Test Loss")
plt.xlabel('Iterations')
plt.ylabel('Miss classificiation error')
plt.title('Iterations vs Miss Classification for Feature k=500')
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.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()


plt.show()
```




Iterations vs Miss Classification for Feature k=500
Train Loss
Test Loss
Miss classificiation error
0.58
0.57
0.56
0.55
0.54
0.53
0.52
0.51
0
100
200
300
400
500
Iterations



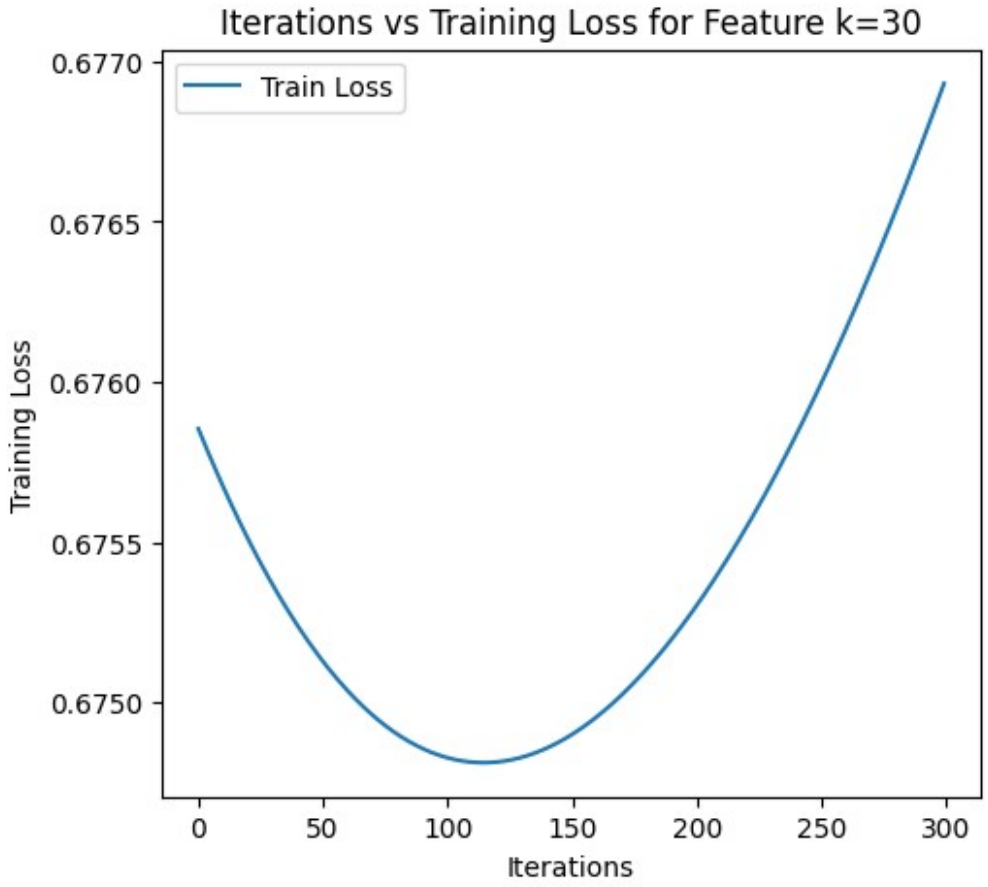

```
results = pd.DataFrame({
    'K-features:': k_features,
    'Train MisClass Error:': train_misclass_errors,
    'Test MisClass Error:': valid_misclass_errors
})

print(results)
```

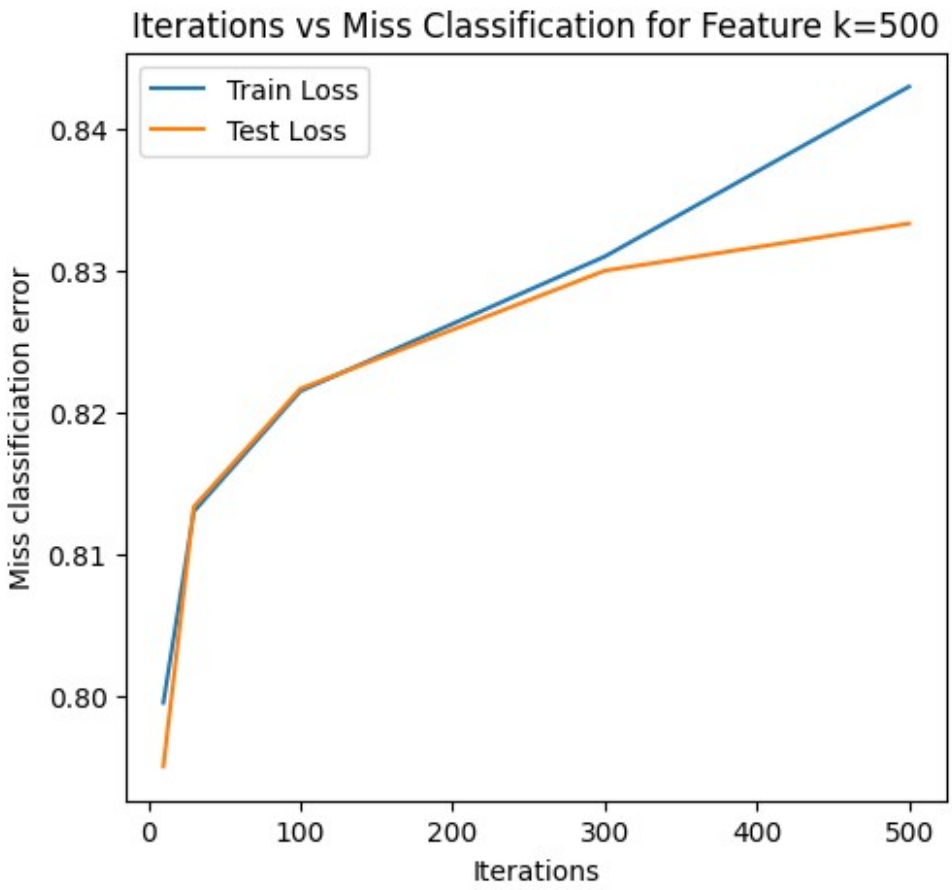
	K-features:	Train MisClass Error:	Test MisClass Error:
0	10	0.536667	0.513333
1	30	0.546667	0.513333
2	100	0.566667	0.516667
3	300	0.576667	0.516667
4	500	0.583333	0.523333

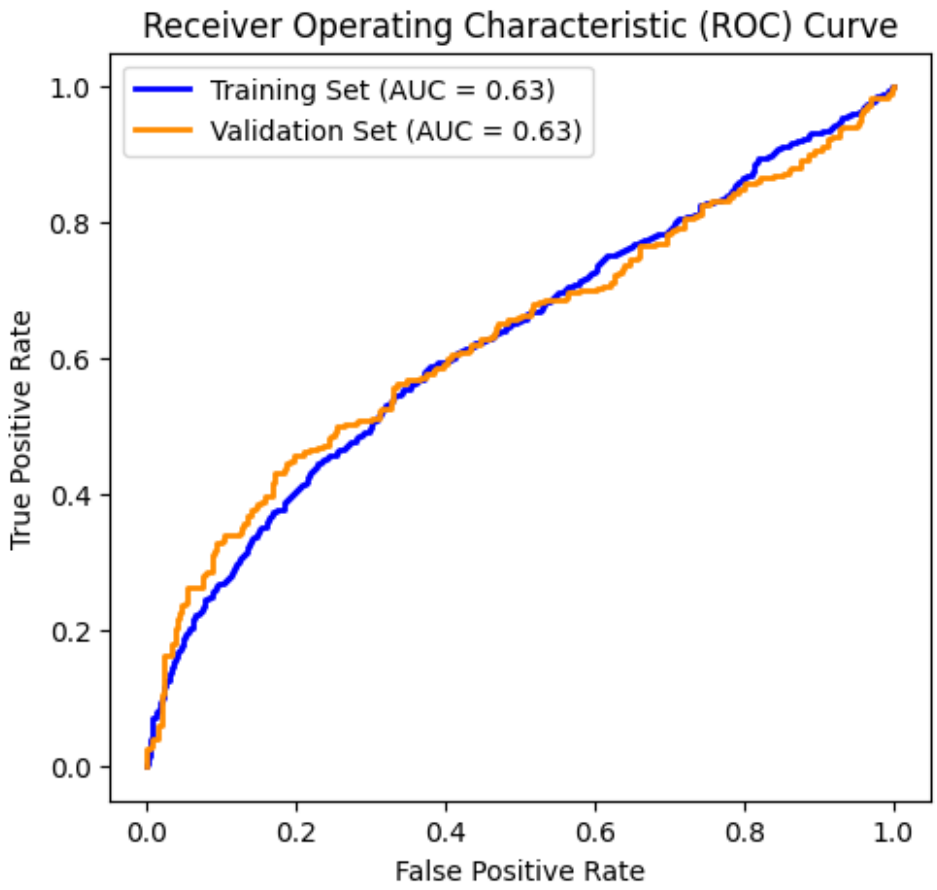
```
# 1c)  MADELON DATASET

X = np.empty(1)
y = np.empty(1)
Xtest = np.empty(1)
yTest = np.empty(1)
probChar = 'c'
```

```
X, y, Xtest, yTest = load_dataset(X, y, Xtest, yTest, probChar)
X, y, Xtest, yTest = normalize(X, y, Xtest, yTest)

iterations = 300
s = 0.0001
mu = 200

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

k_features = [10,30,100,300,500]
lamb = 0.001

train_misclass_errors = []
valid_misclass_errors = []
train_loss_30 = []
fpr_train_list = []
tpr_train_list = []
roc_auc_train_list = []
fpr_valid_list = []
tpr_valid_list = []
roc_auc_valid_list = []

FSA(X, y, Xtest, yTest, w, s, mu, lamb, k_features, iterations)

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(range(iterations), train_loss_30, label="Train Loss")
plt.xlabel('Iterations')
plt.ylabel('Training Loss')
plt.title('Iterations vs Training Loss for Feature k=30')
plt.legend()

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(k_features, train_misclass_errors, label="Train Loss")
plt.plot(k_features, valid_misclass_errors, label="Test Loss")
plt.xlabel('Iterations')
plt.ylabel('Miss classificiation error')
plt.title('Iterations vs Miss Classification for Feature k=500')
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.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

```
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()


plt.show()
```




Iterations vs Miss Classification for Feature k=500
Train Loss
Test Loss
0.84
0.83
0.82
0.81
0.80
Miss classificiation error
0
100
200
300
400
500
Iterations




```
results = pd.DataFrame({
    'K-features:': k_features,
    'Train MisClass Error:': train_misclass_errors,
    'Test MisClass Error:': valid_misclass_errors
})

print(results)
```

	K-features:	Train MisClass Error:	Test MisClass Error:
0	10	0.7995	0.795000
1	30	0.8130	0.813333
2	100	0.8215	0.821667
3	300	0.8310	0.830000
4	500	0.8430	0.833333