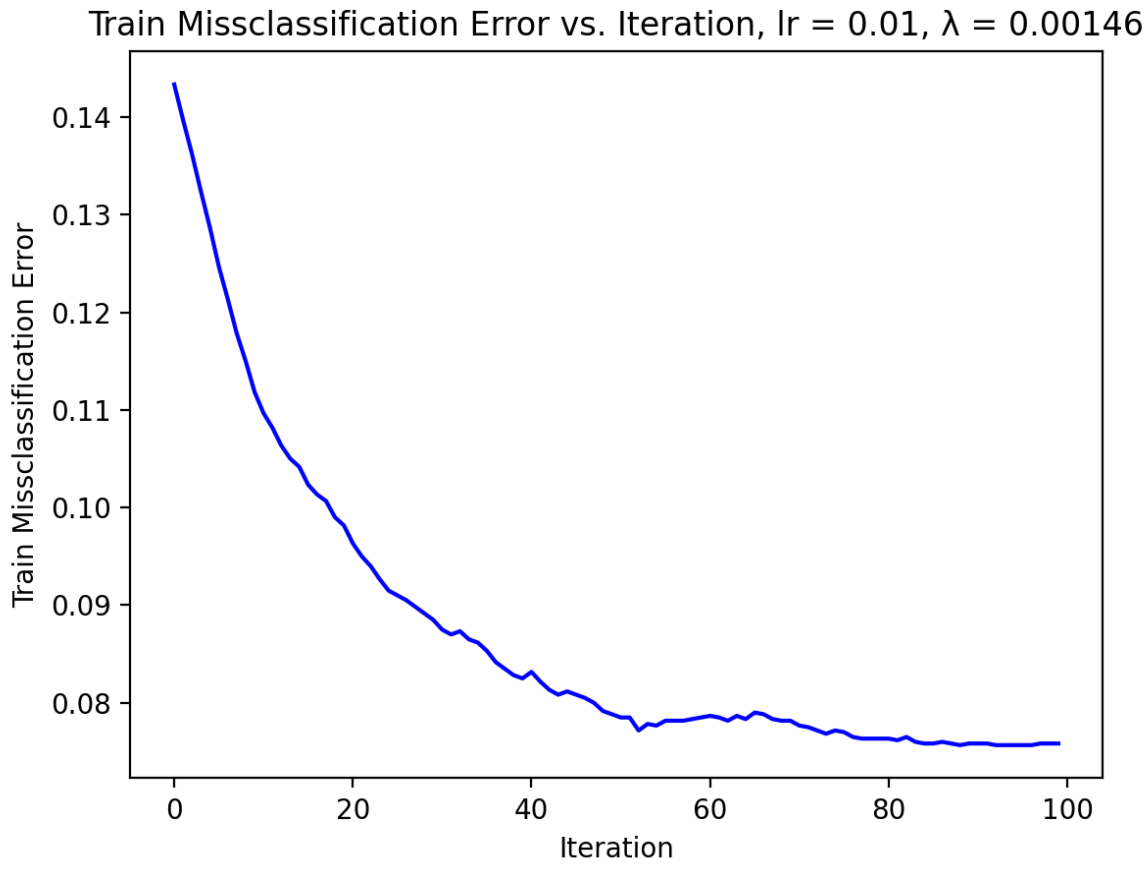
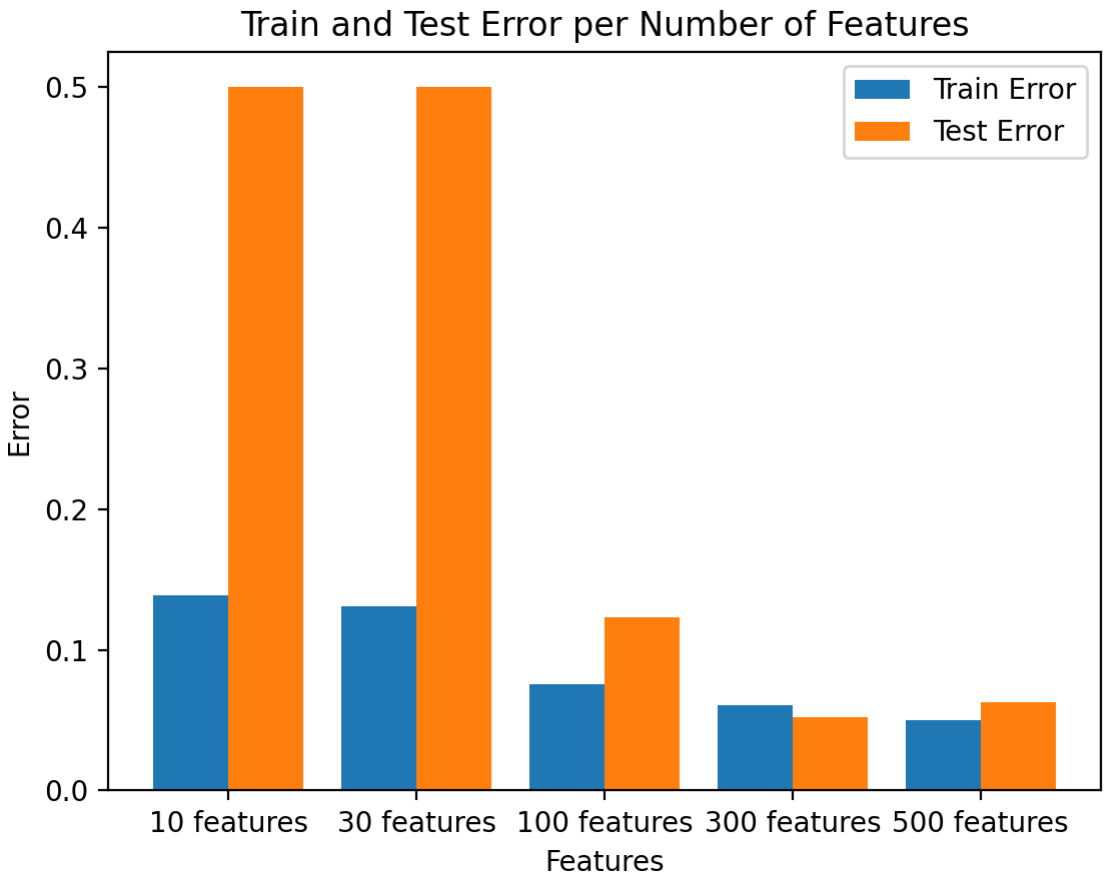
Part A

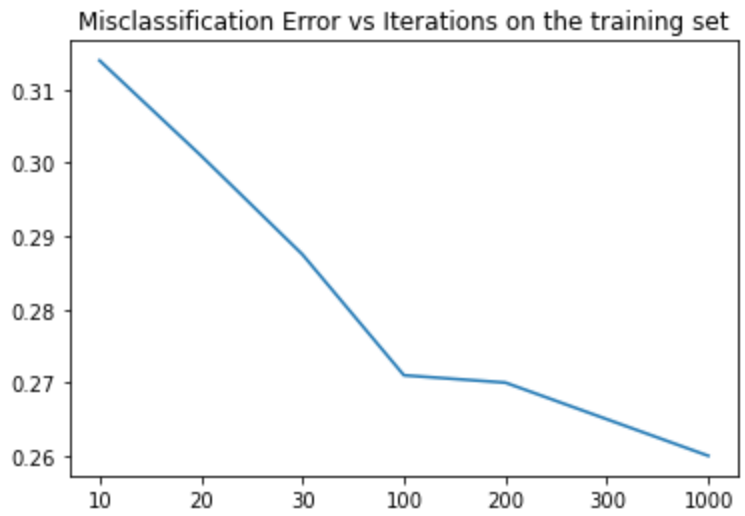
Features (loss = 0.01)	λ	Train Error	Test Error
500	0.00072	5.01%	6.3%
300	0.00092	6.05%	5.2%
100	0.00146	7.58%	12.3%
30	0.00276	13.1%	50%
10	0.00326	13.9%	50%





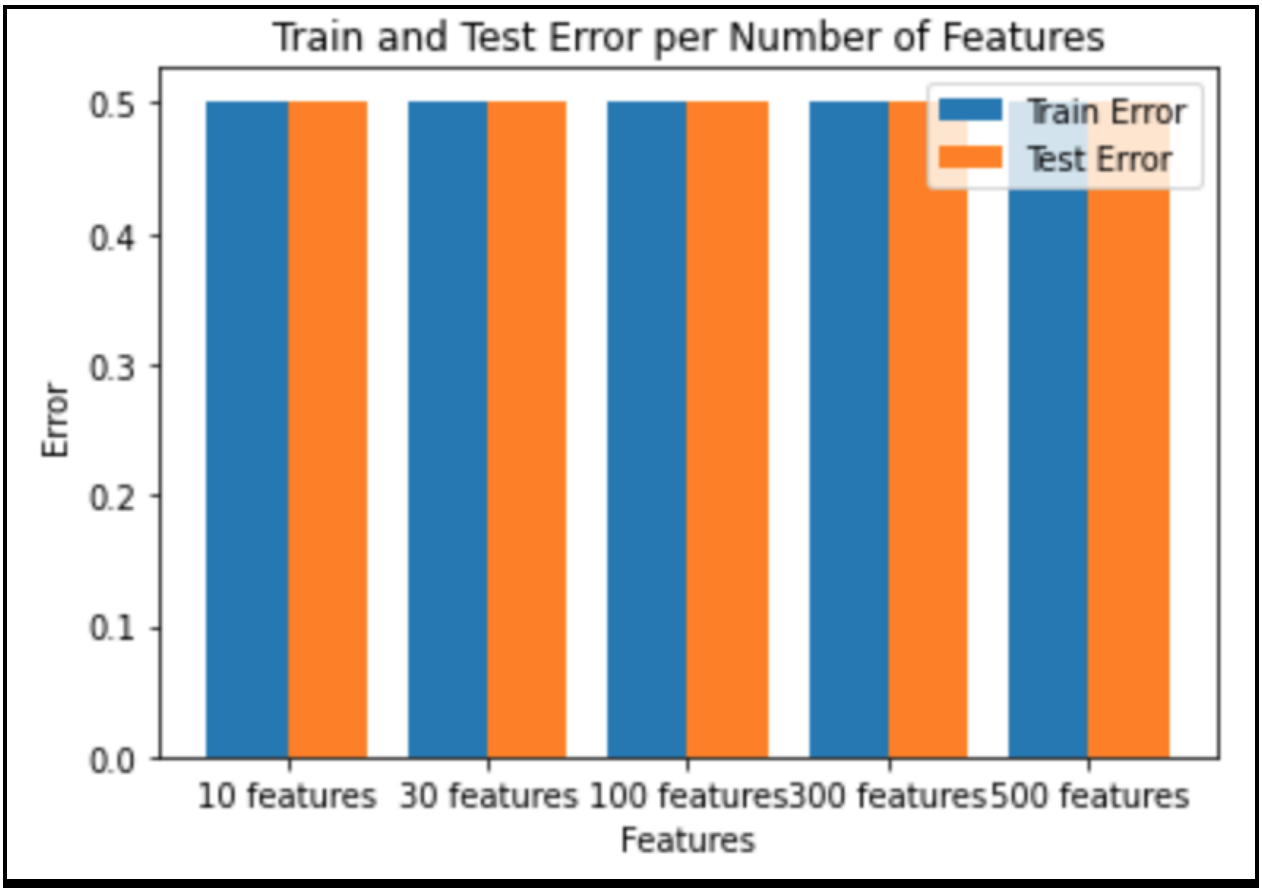
Part B:

Features (eta = 0.01)	λ	Train Error	Test Error
10	.00191	38.95%	50%
30	.001	37.35%	50%
100	.0065	32.75%	50%
300	.003	27.65%	50%
500	.00001	27.1	50%



Part C

Features (loss = 0.01)	λ	Train Error	Test Error
500	.00026499	50.167%	50.167%
300	.000376	50.167%	50.167%
100	.0007469	50.167%	50.167%
30	.001525	50.167%	50.167%
10	.002825	50.167%	50.167%




```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd




METRIC = "Loss"


class TISP:
    def __init__(self,  X, y, learnRate=0, Lambda=.3, iterations=300):
        self.iterations = iterations
        if  (learnRate == 0):
            self.learnRate = 1 / X.shape[1]
        else:
            self.learnRate = learnRate
        self.X = X
```

```
        self.y = y
        self.y0 = np.zeros(y.shape)
        for i, num in enumerate(self.y0):
            if (y[i] == 1):
                self.y0[i] = y[i]
        self.Lambda = Lambda
        self.weights = np.zeros(X.shape[1]) / X.shape[1]
        self.weights = self.weights[..., None]
    def L(self):
        loss = 0
        t0 = np.dot(self.X, self.weights)
        t1 = np.log (1 + np.exp(t0))
        t2 = self.y * t0
        t3 = t2 - t1
        t4 = np.sum(t3) / self.X.shape[1]
        return t4
    def threshold(self):
        for i, num in enumerate(self.weights):
            if (abs(num) <= self.Lambda):
                self.weights[i] = 0
            else:
                self.weights[i] = self.weights[i] #hard penalty
    def gradientAscent(self, plot=False): #trains model, returns loss
        losses = list()
        for i in range(self.iterations):
            t0 = np.dot(self.X, self.weights)
            t1 = t0 / (1 + np.exp( t0 ))
            t2 = self.y0 - t1
            t4 = self.weights + (self.learnRate / self.X.shape[1] *
np.dot(self.X.T , t2))
            self.weights = t4
            self.threshold()
            if (METRIC == "Loss"):
                losses.append(self.L())
            elif (METRIC == "Train Misclassification Error"):
                losses.append(self.error(self.X, self.y))
            print("Iterations: {} {}: {}".format(i, METRIC, losses[i]))
        if (plot == True):
            itrs = range(self.iterations)
            plt.plot(itrs, losses , c='b')
```

```
            plt.xlabel("Iteration")
            plt.ylabel(METRIC)
            plt.title("{} vs. Iteration, lr = {}, λ = {}".format(METRIC,
self.learnRate, self.Lambda))
            plt.show()
        return self.L()
    def error(self, X, y):
        ret = 0
        for j, obj in enumerate(X):
            hyp = 1 / (1 + np.exp(-1 * np.dot(self.weights[:,0], obj)))
            pred = -1
            if (hyp > .5):
                pred = 1
            if (pred != y[j]):
                ret = ret + 1
        return ret / X.shape[0]
    def getFeatures(self):
        ct = 0
        for obj in self.weights:
            if (obj == 0):
                ct = ct + 1
        return self.X.shape[1]  - ct


def loadAnnoyingFile(filename): #loads dexter data
    f = open(filename, "r")
    Lines = f.readlines()
    matrix = np.zeros((len(Lines),20000))
    for i, line in enumerate(Lines):
        mystr = line
        while(mystr != '\n'):
            colonindex = mystr.find(':')
            spaceIndex = mystr.find(' ')
            firstNum = int(mystr[0:colonindex])
            secondNum = int(mystr[colonindex+1:spaceIndex])
            matrix[i, firstNum] = secondNum
            mystr = mystr[spaceIndex+1:]
    return matrix
```

```
trainX = pd.read_csv("gisette_train.data", sep=' ', header=None)
trainX = np.array(trainX)
trainX = trainX[:, 0:5000]
testX = pd.read_csv("gisette_valid.data", sep=' ', header=None)
testX = np.array(testX)
testX = testX[:, 0:5000]
trainy = pd.read_csv("gisette_train.labels", sep=' ', header=None)
trainy = np.array(trainy)
testy = pd.read_csv("gisette_valid.labels", sep=' ', header=None)
testy = np.array(testy)
for i in range(5000):
    if (np.std(trainX[:,i]) != 0):
        trainX[:,i] = (trainX[:,i] - np.mean(trainX[:,i])) /
np.std(trainX[:,i])
        testX[:,i] = (testX[:,i] - np.mean(trainX[:,i])) /
np.std(trainX[:,i])

clf = TISP(trainX, trainy, learnRate=.01, Lambda=.00072, iterations=100)
loss = clf.gradientAscent(plot=True)
print("Features: {}".format(clf.getFeatures()))
print("Train error: {} Test error: {}".format(     clf.error(trainX,
trainy) , clf.error(testX,testy)   ))

#----------------------------------------
#HISTOGRAM
#----------------------------------------
bottom = ["10 features", "30 features", "100 features", "300 features",
"500 features"]
TrainError = [.139, .131, .0758, .0605, .0501]
TestError = [.5, .5, .123, .052, .063]

xAxis = np.arange(len(bottom))

plt.bar(xAxis - 0.2, TrainError, 0.4, label = 'Train Error')
plt.bar(xAxis + 0.2, TestError, 0.4, label = 'Test Error')

plt.xticks(xAxis, bottom)
plt.xlabel("Features")
plt.ylabel("Error")
```

```
plt.title("Train and Test Error per Number of Features")
plt.legend()
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import csv

METRIC = "Loss"


class TISP:
    def __init__(self,  X, y, learnRate=0, Lambda=.3, iterations=300):
        self.iterations = iterations
        if  (learnRate == 0):
            self.learnRate = 1 / X.shape[1]
        else:
            self.learnRate = learnRate
        self.X = X
        self.y = y
        self.y0 = np.zeros(y.shape)
        for i, num in enumerate(self.y0):
            if (y[i] == 1):
                self.y0[i] = y[i]
        self.Lambda = Lambda
        self.weights = np.zeros(X.shape[1]) / X.shape[1]
        self.weights = self.weights[..., None]
    def L(self):
        loss = 0
        t0 = np.dot(self.X, self.weights)
        t1 = np.log (1 + np.exp(t0))
        t2 = self.y * t0
        t3 = t2 - t1
        t4 = np.sum(t3) / self.X.shape[1]
        return t4
    def threshold(self):
        for i, num in enumerate(self.weights):
            if (abs(num) <= self.Lambda):
                self.weights[i] = 0
            else:
                self.weights[i] = self.weights[i] #hard penalty
    def gradientAscent(self, plot=False): #trains model, returns loss
        losses = list()
        for i in range(self.iterations):
```

```
            t0 = np.dot(self.X, self.weights)
            t1 = t0 / (1 + np.exp( t0 ))
            t2 = self.y0 - t1
            t4 = self.weights + (self.learnRate / self.X.shape[1] *
np.dot(self.X.T , t2))
            self.weights = t4
            self.threshold()
            if (METRIC == "Loss"):
                losses.append(self.L())
            elif (METRIC == "Train Misclassification Error"):
                losses.append(self.error(self.X, self.y))
            print("Iterations: {} {}: {}".format(i, METRIC, losses[i]))
        if (plot == True):
            itrs = range(self.iterations)
            plt.plot(itrs, losses , c='b')
            plt.xlabel("Iteration")
            plt.ylabel(METRIC)
            plt.title("{} vs. Iteration, lr = {}, λ = {}".format(METRIC,
self.learnRate, self.Lambda))
            plt.show()
        return self.L()
    def error(self, X, y):
        ret = 0
        for j, obj in enumerate(X):
            hyp = 1 / (1 + np.exp(-1 * np.dot(self.weights[:,0], obj)))
            pred = -1
            if (hyp > .5):
                pred = 1
            if (pred != y[j]):
                ret = ret + 1
        return ret / X.shape[0]
    def getFeatures(self):
        ct = 0
        for obj in self.weights:
            if (obj == 0):
                ct = ct + 1
        return self.X.shape[1]  - ct


trainX = pd.read_csv('dexter_train.data')
trainX = np.array(trainX)
testX = pd.read_csv("dexter_valid.data")
testX = np.array(testX)
trainy = pd.read_csv("dexter_train.labels")
trainy = np.array(trainy)
testy = pd.read_csv("dexter_valid.labels")
testy = np.array(testy)
```

```
trainX_l = []
testX_l = []

for i in range(0,299):
    trainX_d = np.zeros(20000)
    testX_d = np.zeros(20000)


    d1 = trainX[i,0].split(' ')
    for j in range(len(d1)):
        q = d1[j].split(':')
        if len(q) > 1:
            q1 = int(q[0])
            q2 = int(q[1])

            trainX_d[int(q1)] = q2
    trainX_l.append(trainX_d)

    d2 = testX[i,0].split(' ')
    for j in range(len(d2)):
        q = d2[j].split(':')
        if len(q) > 1:
            q1 = int(q[0])
            q2 = int(q[1])

            testX_d[q1] = q2


    testX_l.append(testX_d)

trainX_l = np.array(trainX_l)
testX_l = np.array(testX_l)
trainX_l = np.array(trainX_l).astype(int)
testX_l = np.array(testX_l).astype(int)
trainy = np.array(trainy).astype(int)
testy = np.array(testy).astype(int)



for i in range(299):
    if (np.std(trainX_l[:,i]) != 0):
        trainX_l[:,i] = (trainX_l[:,i] - np.mean(trainX_l[:,i])) /
np.std(trainX_l[:,i])
```

```
        testX_l[:,i] = (testX_l[:,i] - np.mean(trainX_l[:,i])) /
np.std(trainX_l[:,i])

clf = TISP(trainX_l, trainy, learnRate=.01, Lambda=.00026499,
iterations=100)
loss = clf.gradientAscent(plot=True)
print("Features: {}".format(clf.getFeatures()))
print("Train error: {} Test error: {}".format(     clf.error(trainX_l,
trainy) , clf.error(testX_l,testy)   ))

bottom = ["10 features", "30 features", "100 features", "300 features",
"500 features"]
TrainError = [0.50167, 0.50167, 0.50167, 0.50167, 0.50167]
TestError = [0.50167, 0.50167, 0.50167, 0.50167, 0.50167]

xAxis = np.arange(len(bottom))

plt.bar(xAxis - 0.2, TrainError, 0.4, label = 'Train Error')
plt.bar(xAxis + 0.2, TestError, 0.4, label = 'Test Error')

plt.xticks(xAxis, bottom)
plt.xlabel("Features")
plt.xlabel("Features")
plt.ylabel("Error")
plt.title("Train and Test Error per Number of Features")
plt.legend()
plt.show()
```