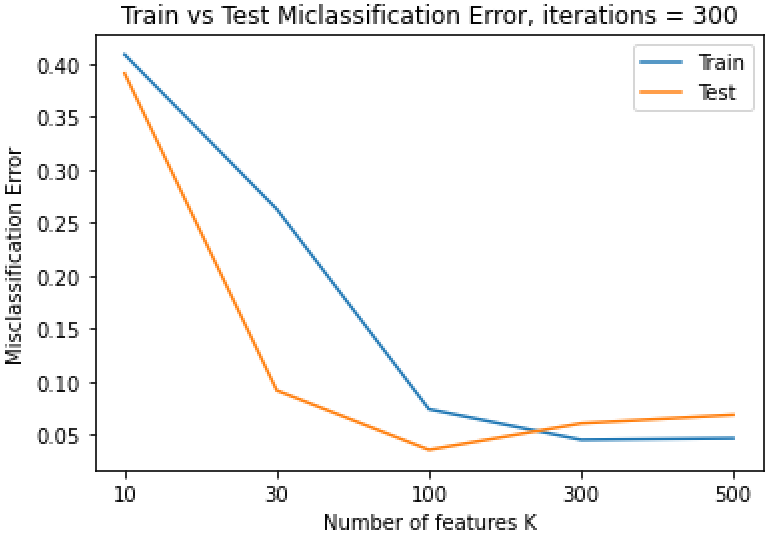
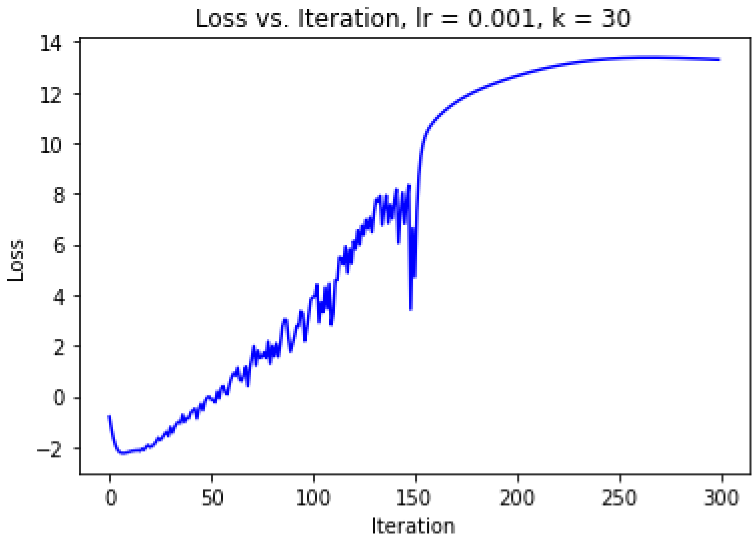
Abelardo Riojas, Jeret McCoy, Gustavo Flores
STA 5635 Spring 2022
Prof. Barbu

Homework 6 Report

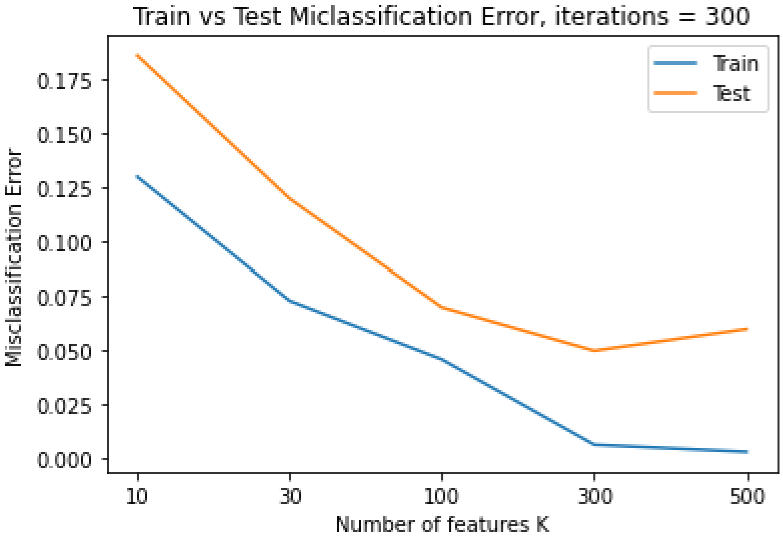
Number of features	Train error	Test error
10	0.40883	0.391
30	0.263	0.092
100	0.0743	0.036
300	0.0455	0.061
500	0.047	0.069

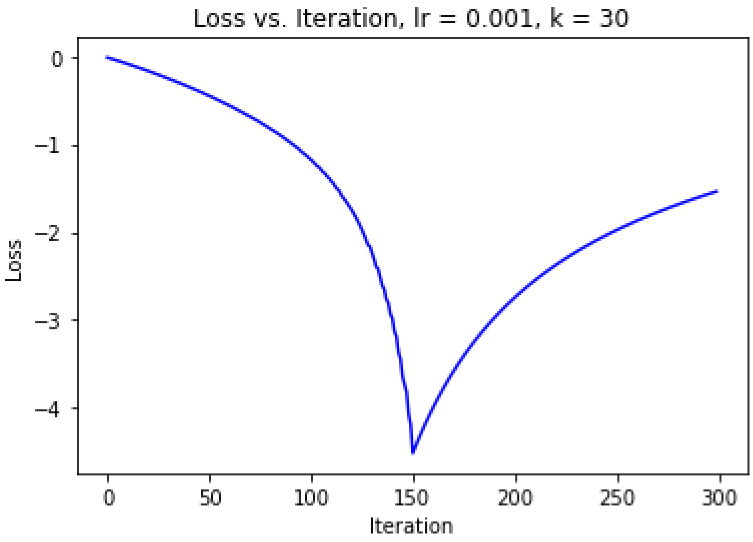




Part B Dexter

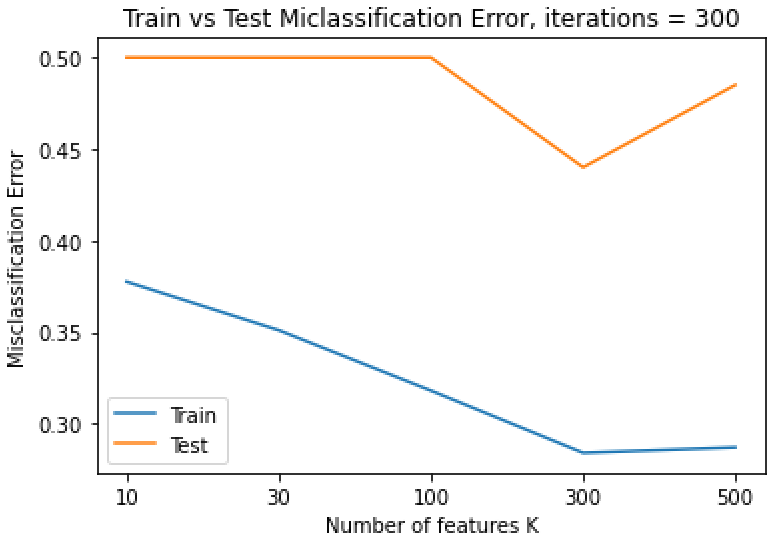
Features	Train Misclass Error	Test Misclass Error
10	0.13	0.186
30	0.073	.12
100	0.046	.07
300	0.0066	.05
500	0.0033	0.06

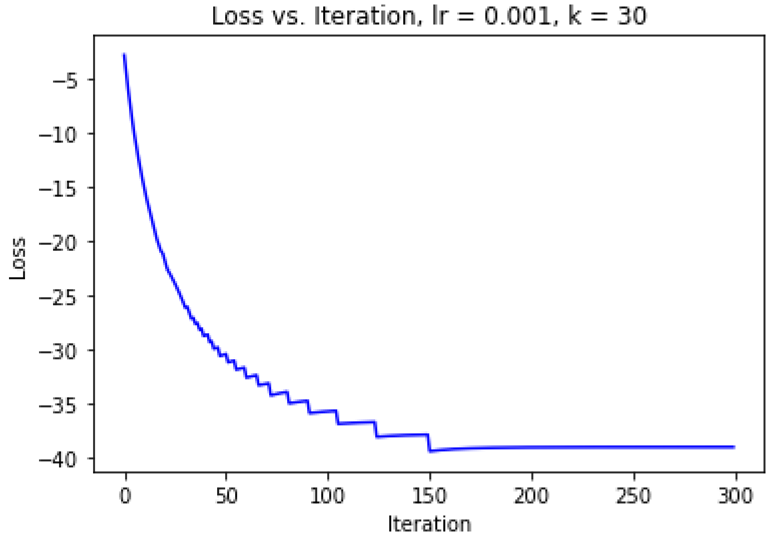




Part C Madelon

Features	Train Misclass Error	Test Misclass Error
10	0.3775	.5
30	0.351	.5
100	0.318	.5
300	0.284	.44
500	0.287	0.485





Code below

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from numpy import linalg as LA


TME = "Train Misclassification Error"
```

```
LOSS = "Loss"
METRIC =  LOSS
FILENAME = "gisette"


class FSA:
    def __init__(self,  X, y, testX, testy, learnRate=0,
k=30,iterations=300, s=0.0001, mu=100):
        self.iterations = iterations
        if  (learnRate == 0):
            self.learnRate = 1 / X.shape[1]
        else:
            self.learnRate = learnRate
        self.X = X
        self.y = y
        self.testX = testX
        self.testy = testy
        self.y0 = np.zeros(y.shape)
        for i, num in enumerate(self.y0):
            if (y[i] == 1):
                self.y0[i] = y[i]
        self.k = k
        self.s = s
        self.mu = mu
        self.M = X.shape[1]
        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])
        t4 = t4 + (self.s * LA.norm(self.weights[:,0]))
        return t4
    def threshold(self, itr):
        t0 = (self.iterations - (2 * itr))/(2*itr*self.mu +
self.iterations)
        t0 = max(0, t0)
        t0 = (self.M - self.k) * t0
        newM = self.k + t0
        t1 = np.abs(self.weights[:,0])
        indexes = np.argsort(t1)
        t2 = int(self.weights.shape[0] - newM)
        if (t2 == 0):
            return self.weights
        indexes = indexes[0:t2]   #verified that this removes lowest
        self.weights = np.delete(self.weights, indexes, 0)

        self.X = np.delete(self.X, indexes, 1)
        self.testX = np.delete(self.testX, indexes, 1)


    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))
            t4 = t4 + (2 * self.s * self.weights)
            self.weights = t4
            if (i > 0):
                self.threshold(i)
            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 = {}, k = {}".format(METRIC,
self.learnRate, self.k))
            plt.show()
        return self.L()
    def trainError(self):
        ret = 0
        for j, obj in enumerate(self.X):
            hyp = 1 / (1 + np.exp(-1 * np.dot(self.weights[:,0], obj)))
            pred = -1
            if (hyp > .5):
                pred = 1
            if (pred != self.y[j]):
                ret = ret + 1
        return ret / self.X.shape[0]
    def testError(self):
        ret = 0
        for j, obj in enumerate(self.testX):
            hyp = 1 / (1 + np.exp(-1 * np.dot(self.weights[:,0], obj)))
            pred = -1
            if (hyp > .5):
                pred = 1
            if (pred != self.testy[j]):
                ret = ret + 1
        return ret / self.testX.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


if (FILENAME == "gisette"):
    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)

if (FILENAME == "dexter"):
    trainX = loadAnnoyingFile("dexter_train.data")
    trainy = pd.read_csv("dexter_train.labels", sep=' ', header=None)
    testX = loadAnnoyingFile("dexter_valid.data")
    testy = pd.read_csv("dexter_valid.labels", sep=' ', header=None)
    trainy = np.array(trainy)
    testy = np.array(testy)

if (FILENAME == "madelon"):
    trainX = pd.read_csv("madelon_train.data", sep=' ', header=None)
    trainX = np.array(trainX)
    trainX = trainX[:, 0:trainX.shape[1]-1]
    testX = pd.read_csv("madelon_valid.data", sep=' ', header=None)
    testX = np.array(testX)
    testX = testX[:, 0:testX.shape[1]-1]
    trainy = pd.read_csv("madelon_train.labels", sep=' ', header=None)
    trainy = np.array(trainy)
    testy = pd.read_csv("madelon_valid.labels", sep=' ', header=None)
    testy = np.array(testy)

for i in range(trainX.shape[1]):
    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 = FSA(trainX, trainy, testX, testy, learnRate=0.001, k=100,
iterations=300)
loss = clf.gradientAscent(plot=True)
print("Features: {}".format(clf.getFeatures()))
print("Train error: {} Test error: {}".format(     clf.trainError() ,
clf.testError()  ))
```