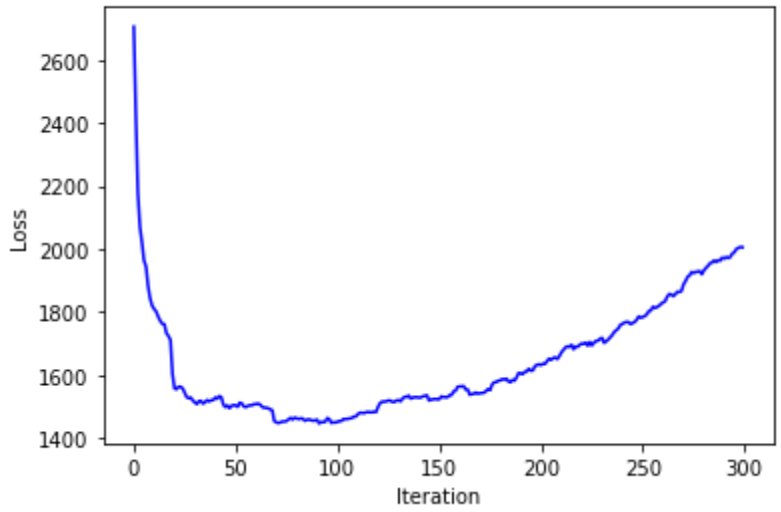
Abelardo Riojas, Gustavo Flores, and Jeret McCoy
STA 5635 Spring 2022
Prof. Barbu

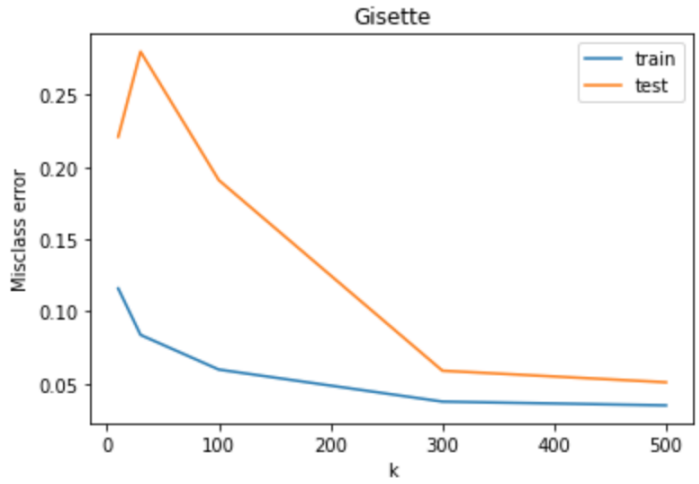
Homework 7 Report

Part A: Gisette

K	Train Misclass Error	Test Misclass Error
10	0.116	0.221
30	0.0838	0.28
100	0.0598	0.191
300	0.0376	0.059
500	0.0335	0.051

k=300

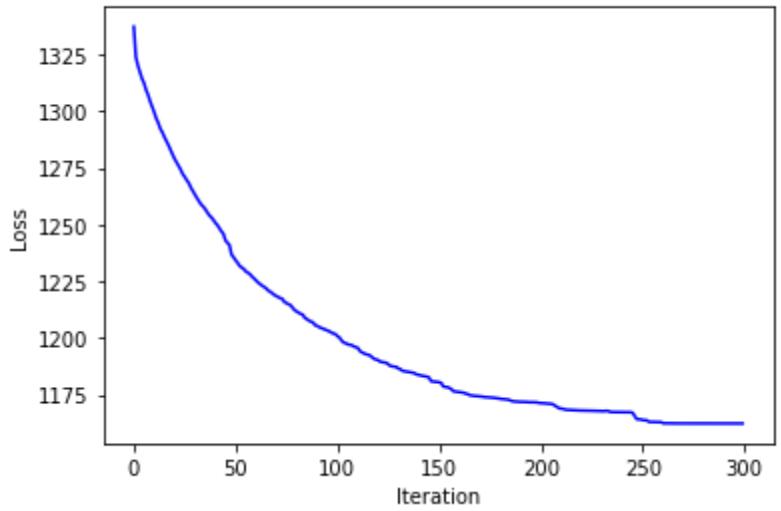


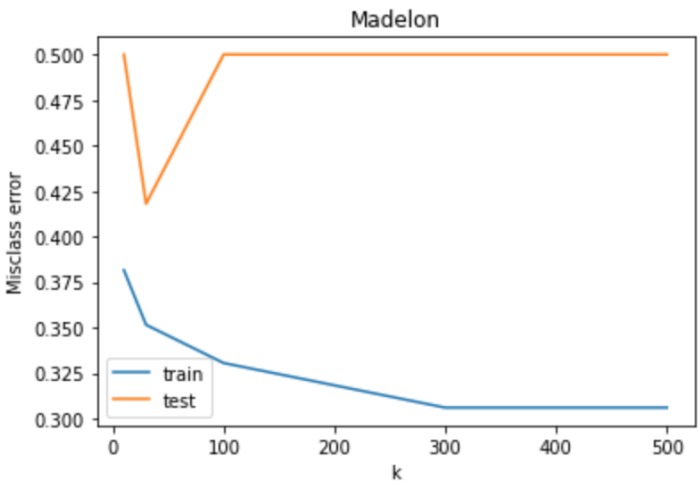


Part B: Madelon

K	Train Misclass Error	Test Misclass Error
10	0.3815	0.5
30	0.3515	0.418
100	0.3305	0.5
300	0.306	0.5
500	0.306	.5

k=300

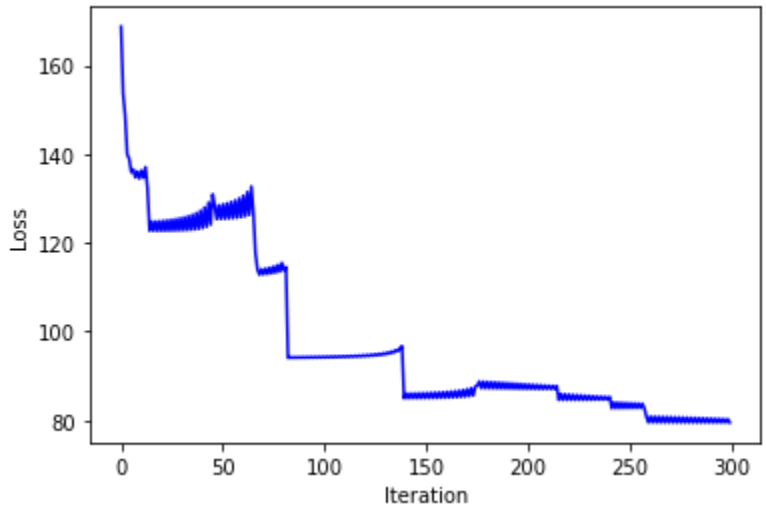


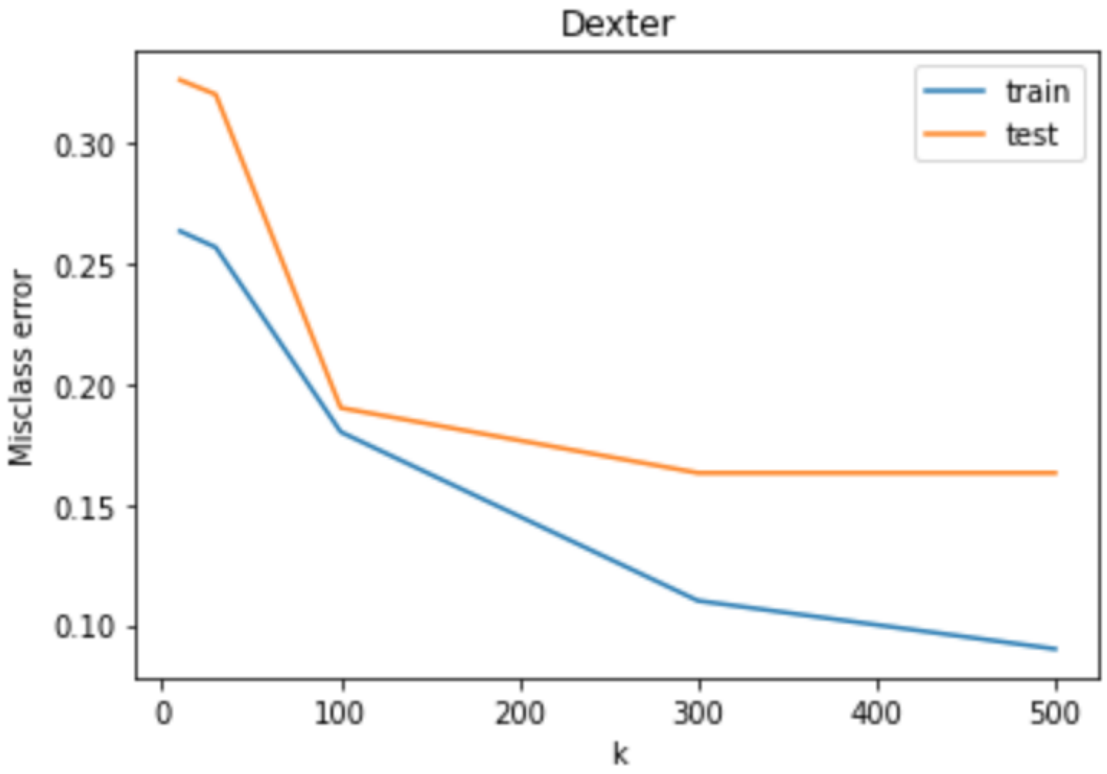


Part C: Dexter

K	Train Misclass Error	Test Misclass Error
10	0.2633	0.326
30	0.2566	0.320
100	0.180	0.190
300	0.110	0.163
500	0.09	0.163

k=300




Dexter
train
test
0.30
0.25
0.20
0.15
0.10
Misclass error
0
100
200
300
400
500
k

Code below:

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

from sklearn.metrics import mean_squared_error

TME = "Train Misclassification Error"
LOSS = "Loss"
METRIC =  LOSS
FILENAME = "dexter"


class LogitBoost:
   def __init__(self,  X, y, testX, testy, iterations=300):
      self.iterations = iterations
      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.F = np.zeros(X.shape[1])
      self.F = self.F[..., None]
   def L(self):
      t0 = np.dot(self.X, self.F)
      t1 = self.y * t0
      t2 = np.log (1 + np.exp(-1 * t1))
      t3 = (np.sum(t2))
      return t3
   def WLS(self, weights, z):
      t0 = self.X * weights
      top = t0 * z
      top = np.sum(top, axis=0)
      bottom = t0 * self.X
      bottom = np.sum(bottom, axis=0)
      return top / (bottom + 1e-5)

   def boost(self, plot=False): #trains model, returns loss
```

```
      losses = list()
      featureList = list()
      bestFeature = 0
      for i in range(self.iterations):
            h = np.dot(self.X, self.F)
            t0 = np.exp(-2 * h)
            p = 1 / (t0 + 1)
            weights = np.multiply(p, 1-p)
            t0 = self.y0 - p
            z = t0 / (weights + 1e-6)
            fm = LinearRegression()
            testF = np.zeros(self.X.shape[1])
            testF = testF[..., None]
            testF = self.WLS(weights, z)
            testF = testF[..., None]
            l0 = (self.X.T * testF).T
            l1 = l0 * self.y
            l1 = np.clip(l1, -500, 500)
            l2 = np.log(1 + np.exp(-1 * l1))
            l3 = np.sum(l2, axis=0) #sum every column
            bestFeature = l3.argmin()
            self.F[bestFeature,0] = self.F[bestFeature,0] +
testF[bestFeature,0]
            if (METRIC == "Loss"):
            losses.append(self.L())
            elif (METRIC == "Train Misclassification Error"):
            losses.append(self.trainError())
            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, 0,
0))
            plt.show()
      return self.L()
   def trainError(self):
      ret = 0
      for j, obj in enumerate(self.X):
            hyp = 1 / (1 + np.exp(-2 * np.dot(self.F[:,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):
            self.F = np.clip(self.F, -500, 500)
            t0 = -2 * np.dot(self.F[:,0], obj)
            t0 = np.clip (t0, -500, 500)
            hyp = 1 / (1 + np.exp(t0))
            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.F:
            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 = LogitBoost(trainX, trainy, testX, testy,  iterations=300)
loss = clf.boost(plot=True)
print("Features: {}".format(clf.getFeatures()))
print("Train error: {} Test error: {}".format(  clf.trainError() ,
clf.testError()  ))
```