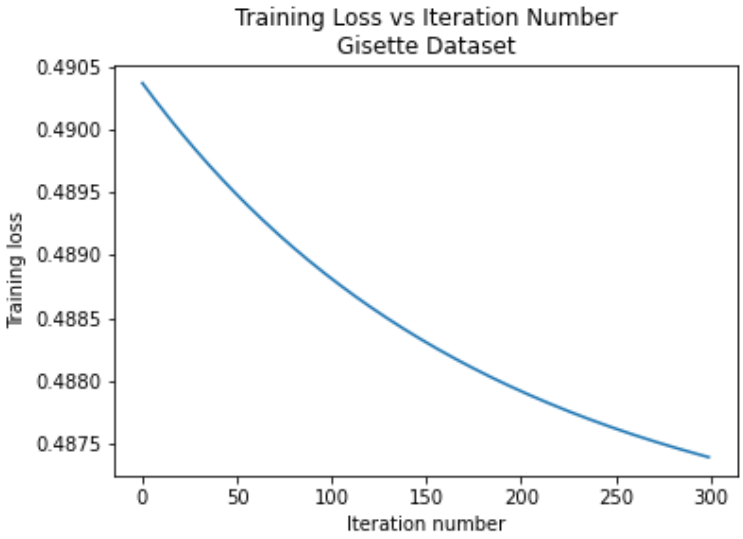
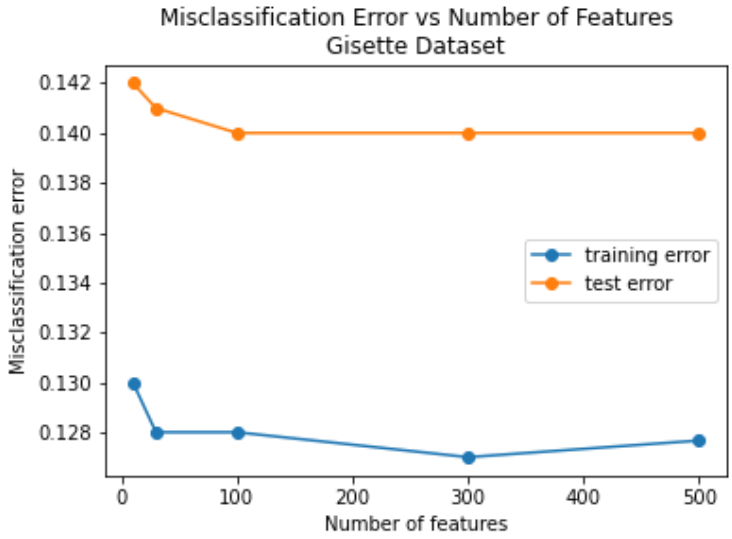
Homework 6

Jarod Klion

February 16th, 2022

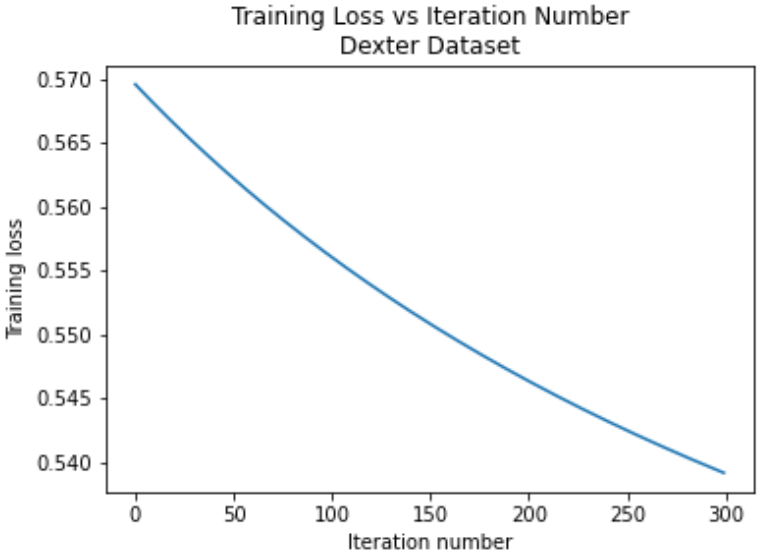
1. Gisette Dataset

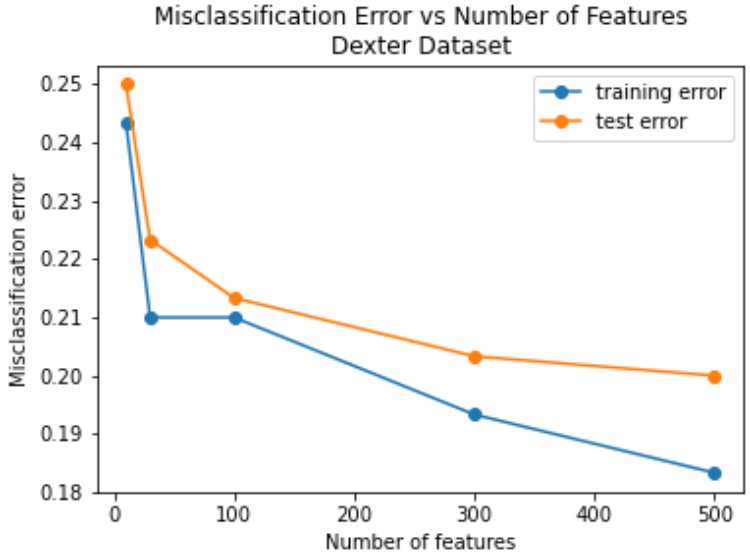




K	Train Error	Test Error
10	0.13	0.142
30	0.128	0.141
100	0.128	0.14
300	0.127	0.14
500	0.127667	0.14

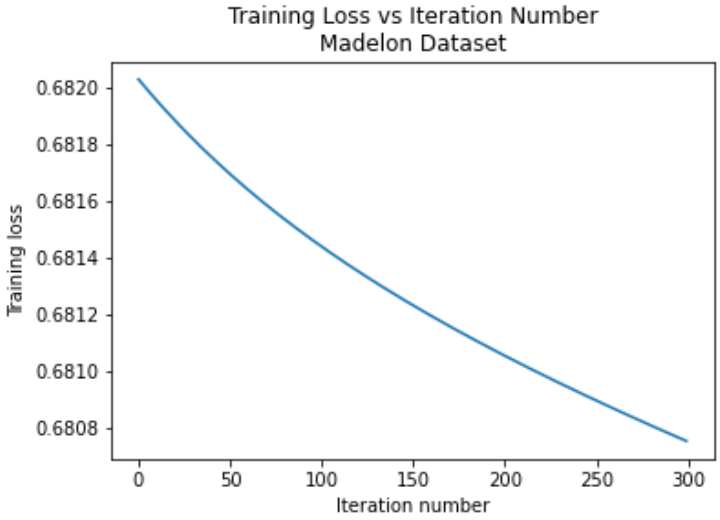
2. Dexter Dataset

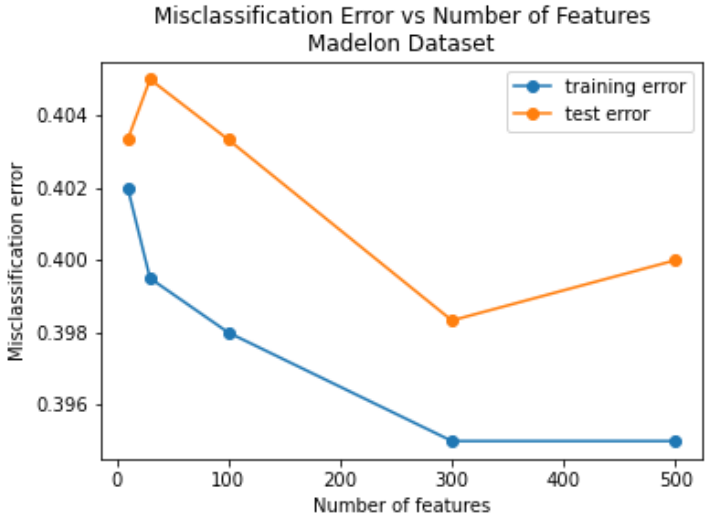




K	Train Error	Test Error
10	0.243333	0.25
30	0.21	0.223333
100	0.21	0.213333
300	0.193333	0.203333
500	0.183333	0.2

3. Madelon dataset





K	Train Error	Test Error
10	0.402	0.403333
30	0.3995	0.405
100	0.398	0.403333
300	0.395	0.398333
500	0.395	0.4

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler


# # Problem 4.a: Gisette dataset

# In[309]:


# Define the FSA class with usual functions
class FSAClassifier:
    def __init__(self, s = 0.0001, mu = 100, eta = 0.1, iterations
= 300):

        #initialize parameters
        self.iterations = iterations
        self.s = s
        self.mu = mu
        self.eta = eta/5000

        self.desired_features = [10, 30, 100, 300, 500] #number of
feature selections wanted
        self.train_errs = []
        self.test_errs = []
        self.k30_train_loss = []


    def sigmoid(self, z):
        '''
        Sigmoid function

        Parameters:
        x : int, float or numpy array

        Returns:
        the sigmoid function applied to each element in the array
        '''
```

```
        return 1 / (1 + np.exp(-z))


    def log_likelihood(self, x, y):
        '''
        Log likelihood of penalized logistic regression FSA for
classification

        Parameters:
        x : regressor matrix, X
        beta: sparsity constraints (array)
        y : labels {0, 1}
        '''
        z = np.dot(x, self.betas)
        return (1 / x.shape[0])*sum(np.log(1 + np.exp(-y * z)))


    def logistic_loss(self, x, y):
        z = np.dot(x, self.betas)
        return (1 / x.shape[0]) * sum(np.log(1 + np.exp(-(y *
z)))) + self.s * np.linalg.norm(self.betas)

    def fit(self, X, y, X_test, y_test):
        '''
        Fit data to the FSA model

        Parameters:
        X_ : 2D Regressor matrix (num_obs, num_feats)
        y_ : 1D array of labels {0, 1}

        Returns:
        training loss, training errors, test errors
        '''
        num_obs = X.shape[0]
        num_feats = X.shape[1]

        #start with b0 = 0
        self.betas = np.zeros(num_feats)

        #loop through each desired number of features
        for k in self.desired_features:
            #loop for num of iterations
            for j in range(self.iterations):

                z = np.dot(X, self.betas)
                y_pred = self.sigmoid(z)
```

```
                #calculate gradient
                gradient = np.dot(X.T, y - y_pred)

                #update betas
                self.betas = self.betas + self.eta * gradient

                #do feature selection
                Mi = k + (num_feats - k) * int(max(0,
(self.iterations - 2 * j)/(2 * j * self.mu + self.iterations)))
                sorted_betas = np.argsort(np.abs(self.betas))
#argsort to get sliced lists
                sorted_betas = sorted_betas[-Mi:]
                #get the last Mi betas
                self.betas = self.betas[sorted_betas]
                X = X[:, sorted_betas]
                X_test = X_test[:, sorted_betas]

                #Calculate training loss for k=30 specifically
                if k == 30:

self.k30_train_loss.append(self.logistic_loss(X, y))

                #Capture the iteration loop's last value
                if (j == self.iterations - 1):
                    self.train_errs.append(1 - self.score(X,y))
                    self.test_errs.append(1 - self.score(X_test,
y_test))


    def predict_proba(self, X):

        z = np.dot(X, self.betas)
        probabilities = self.sigmoid(z)

        return probabilities


    def predict(self, X, threshold=0.5):
        # Thresholding probability to predict binary values
        binary_predictions = np.array(list(map(lambda x: 1 if x >
threshold else 0, self.predict_proba(X))))

        return binary_predictions
```

```
    def score(self, X, y_true):
        #classification accuracy
        y_pred = self.predict(X)
        acc = accuracy_score(y_true, y_pred)

        return acc


# In[304]:


gis_train = pd.read_csv("../datasets/Gisette/gisette_train.data",
sep = ' ', header=None).dropna(axis=1)
gis_train_labels =
np.where(np.ravel(pd.read_csv("../datasets/Gisette/gisette_train.l
abels", sep = ' ', header=None).values) == -1, 0, 1)
gis_test = pd.read_csv("../datasets/Gisette/gisette_valid.data",
sep = ' ', header=None).dropna(axis=1)
gis_test_labels =
np.where(np.ravel(pd.read_csv("../datasets/Gisette/gisette_valid.l
abels", sep = ' ', header=None).values) == -1, 0, 1)


# In[305]:


#Normalize training data in gisette to have mean 0 and standard
deviation 1
sc_gis = StandardScaler()
sc_gis.fit(gis_train)
gis_train_norm = sc_gis.transform(gis_train)
#apply the same transformation to testing data
gis_test_norm = sc_gis.transform(gis_test)


# In[310]:


gis_model = FSAClassifier()
gis_model.fit(gis_train_norm, gis_train_labels, gis_test_norm,
gis_test_labels)


# In[349]:
```

```
plt.figure(figsize=(13,4))
plt.subplot(121)
plt.plot(range(300), gis_model.k30_train_loss)
plt.xlabel("Iteration number")
plt.ylabel("Training loss")
plt.title("Training Loss vs Iteration Number\nGisette Dataset")

plt.subplot(122)
plt.plot(gis_model.desired_features, gis_model.train_errs, '-o',
label = "training error")
plt.plot(gis_model.desired_features, gis_model.test_errs, '-o',
label = "test error")
plt.xlabel("Number of features")
plt.ylabel("Misclassification error")
plt.title("Misclassification Error vs Number of Features\nGisette
Dataset")
plt.legend()

plt.savefig("Gisette.png")
plt.show()


# In[354]:


gis_info = {"K": gis_model.desired_features,
            "Train Error": gis_model.train_errs,
            "Test Error": gis_model.test_errs,
            }
gis_df = pd.DataFrame(gis_info)
gis_df.sort_values(['K'], ascending=True)


# # Problem 4.b: Dexter dataset

# In[314]:


#Read in the data for dexter dataset
dex_train = pd.read_csv("../datasets/dexter/dexter_train.csv",
header=None).dropna(axis=1)
dex_train_labels =
np.where(np.ravel(pd.read_csv("../datasets/dexter/dexter_train.lab
els", sep = ' ', header=None).values) == -1, 0, 1)
dex_test = pd.read_csv("../datasets/dexter/dexter_valid.csv",
header=None).dropna(axis=1)
```

```
dex_test_labels =
np.where(np.ravel(pd.read_csv("../datasets/dexter/dexter_valid.lab
els", sep = ' ', header=None).values) == -1, 0, 1)


# In[315]:


#Normalize training data in dexter to have mean 0 and standard
deviation 1
sc_dex = StandardScaler()
sc_dex.fit(dex_train)
dex_train_norm = sc_dex.transform(dex_train)
#apply the same transformation to testing data
dex_test_norm = sc_dex.transform(dex_test)


# In[316]:


dex_model = FSAClassifier()
dex_model.fit(dex_train_norm, dex_train_labels, dex_test_norm,
dex_test_labels)


# In[350]:


plt.figure(figsize=(13,4))
plt.subplot(121)
plt.plot(range(300), dex_model.k30_train_loss)
plt.xlabel("Iteration number")
plt.ylabel("Training loss")
plt.title("Training Loss vs Iteration Number\nDexter Dataset")

plt.subplot(122)
plt.plot(dex_model.desired_features, dex_model.train_errs, '-o',
label = "training error")
plt.plot(dex_model.desired_features, dex_model.test_errs, '-o',
label = "test error")
plt.xlabel("Number of features")
plt.ylabel("Misclassification error")
plt.title("Misclassification Error vs Number of Features\nDexter
Dataset")
plt.legend()
```

```
plt.savefig("Dexter.png")
plt.show()


# In[353]:


dex_info = {"K": dex_model.desired_features,
            "Train Error": dex_model.train_errs,
            "Test Error": dex_model.test_errs,
            }
dex_df = pd.DataFrame(dex_info)
dex_df.sort_values(['K'], ascending=True)


# # Problem 4.c: Madelon dataset

# In[330]:


mad_train = pd.read_csv("../datasets/madelon/madelon_train.data",
sep=' ', header=None).dropna(axis=1)
mad_train_labels =
np.where(np.ravel(pd.read_csv("../datasets/madelon/madelon_train.l
abels", sep=' ', header=None).dropna(axis=1).values) == -1, 0, 1)
mad_test = pd.read_csv("../datasets/madelon/madelon_valid.data",
sep = ' ', header=None).dropna(axis=1)
mad_test_labels =
np.where(np.ravel(pd.read_csv("../datasets/madelon/madelon_valid.l
abels", sep=' ', header=None).dropna(axis=1).values) == -1, 0, 1)


# In[332]:


#Normalize training data in madelon to have mean 0 and standard
deviation 1
sc_mad = StandardScaler()
sc_mad.fit(mad_train)
mad_train_norm = sc_mad.transform(mad_train)
#apply the same transformation to testing data
mad_test_norm = sc_mad.transform(mad_test)


# In[339]:
```

```
mad_model = FSAClassifier(eta = 0.01)
mad_model.fit(mad_train_norm, mad_train_labels, mad_test_norm,
mad_test_labels)


# In[351]:


plt.figure(figsize=(13,4))
plt.subplot(121)
plt.plot(range(300), mad_model.k30_train_loss)
plt.xlabel("Iteration number")
plt.ylabel("Training loss")
plt.title("Training Loss vs Iteration Number\nMadelon Dataset")

plt.subplot(122)
plt.plot(mad_model.desired_features, mad_model.train_errs, '-o',
label = "training error")
plt.plot(mad_model.desired_features, mad_model.test_errs, '-o',
label = "test error")
plt.xlabel("Number of features")
plt.ylabel("Misclassification error")
plt.title("Misclassification Error vs Number of Features\nMadelon
Dataset")
plt.legend()

plt.savefig("Madelon.png")
plt.show()


# In[352]:


mad_info = {"K": mad_model.desired_features,
            "Train Error": mad_model.train_errs,
            "Test Error": mad_model.test_errs,
            }
mad_df = pd.DataFrame(mad_info)
mad_df.sort_values(['K'], ascending=True)


# In[ ]:
```