Applied Machine Learning
Homework 6

Daniel Gonzalez
Colton Piper
25th of October, 2018

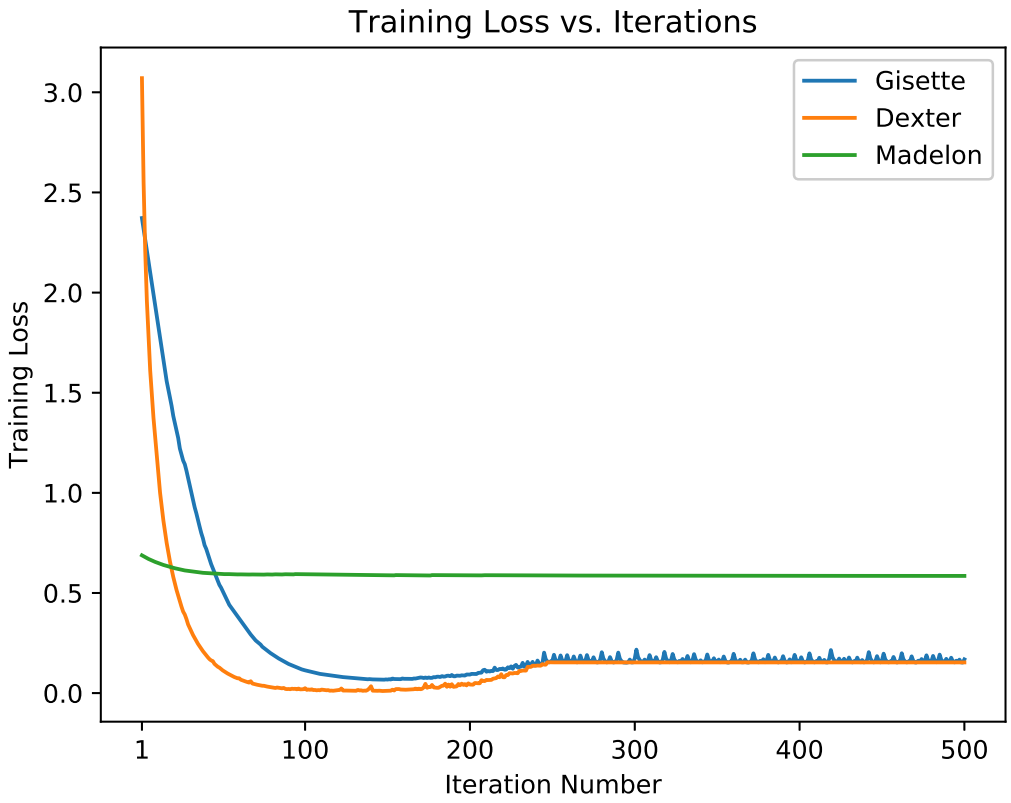
1 Results

1.1 Table

Table 1: Summary of Results: Original Learning Rates

Data Set	Learning Rate η	Features Selected k	Training Error	Test Error
`gisette`	5	10	7.65%	8.30%
		30	3.28%	4.20%
		100	2.60%	3.00%
		300	2.87%	3.90%
`dexter`	10	10	7.00%	12.67%
		30	0.67%	14.33%
		100	0.00%	9.00%
		300	0.00%	11.00%
`madelon`	0.01	10	38.00%	39.50%
		30	35.80%	40.83%
		100	31.20%	40.83%
		300	25.90%	42.83%

1.2 Figures



2 Appendix: Code

If the code looks too small, please zoom in on the pdf. The screenshots are `.png` images, so you should be able to zoom in and read at whatever is a comfortable size for you. The first two screenshots are of the actual FSA code, while the last screenshot is the python code for normalizing and pre-processing the data.

```python
# Daniel Gonzalez, FSU Mathematics PhD
# Colton Piper, FSU Mathematics PhD
# Applied Machine Learning Assignment 6

import re
import numpy as np
import matplotlib.pyplot as plt

training_data = ["./data_norm/gisette/gisette_train.data.npy", "./data_norm/dexter/dexter_train.csv.npy", "./data_norm/madelon/madelon_train.data.npy"]
test_data = ["./data_norm/gisette/gisette_valid.data.npy", "./data_norm/dexter/dexter_valid.csv.npy", "./data_norm/madelon/madelon_valid.data.npy"]

training_labels = ["./data_norm/gisette/gisette_train.labels.npy", "./data_norm/dexter/dexter_train.labels.npy", "./data_norm/madelon/madelon_train.labels.npy"]
test_labels = ["./data_norm/gisette/gisette_valid.labels.npy", "./data_norm/dexter/dexter_valid.labels.npy", "./data_norm/madelon/madelon_valid.labels.npy"]

#SHRINKAGE FACTOR
s = 0.001
#ANNEALING PARAMETER
μ = 100
#NUMBER OF ITERATIONS
N_itr = 500
#LEARNING RATES FOR DIFFERENT DATA SETS
η_master = [5, 10, 0.01]
#VARIABLE SELECTION PARAMETER
K = [10, 30, 100, 300]

#LOSS FUNCTION
def lorenz_loss(X, Y, β):
    z = Y * np.dot(X, β)
    loss = np.log(1 + (z - 1)**2)
    loss[z > 1] = 0
    return np.sum(loss)/X.shape[0] + s * (np.linalg.norm(β)**2)

#GRADIENT OF LORENZ LOSS
def gradient(X, Y, β):
    ybx = Y * np.dot(X, β)
    f = (2*(ybx - 1)/(1 + (ybx - 1)**2))
    f[ybx > 1] = 0
    return np.dot(X.T, f * Y)/X.shape[0] + 2*s*β

#VARIABLE ELIMINATION: RETAIN ONLY THE Mi FEATURES WITH THE LARGEST L2 NORM
def feature_selection(β, Mi):
    sorted_features = np.flip(np.sort(β**2, kind="mergesort", axis=None))
    threshold = sorted_features[Mi-1]
    count = len([f for f in sorted_features[Mi:] if f == threshold])

    β[β**2 < threshold] = 0
    for i in range(0, β.shape[0]):
        if β[i]**2 == threshold and count > 0:
            β[i] = 0
            count -= 1
        elif count <= 0:
            break
    return β

#FEATURE SELECTION WITH ANNEALING: GRADIENT DESCENT FOLLOWED BY VARIABLE ELIMINATION
def fsa(X, Y, validation, η, k):
    N = X.shape[0]
    M = X.shape[1]
    β = np.zeros(M);
    loss = []
    for i in range(1, N_itr+1):
        β = β - η * gradient(X, Y, β)

        #FIND THE NUMBER OF FEATURES TO RETAIN
        Mi = int(k + (M - k)*max(0, (N_itr - 2*i)/(2*i*μ + N_itr)))
        #Mi = k + (M - k)*max(0, int((N_itr - 2*i)/(2*i*μ + N_itr)))
        β = feature_selection(β, Mi)
        indices = np.where(β == 0)

        #DELETE THE IRRELEVANT FEATURES FROM THE WEIGHTS AND DATA
        β = np.delete(β, indices)
```

```
#FEATURE SELECTION WITH ANNEALING: GRADIENT DESCENT FOLLOWED BY VARIABLE ELIMINATION
def fsa(X, Y, validation, η, k):
    N = X.shape[0]
    M = X.shape[1]
    β = np.zeros(M);
    loss = []
    for i in range(1, N_itr+1):
        β = β - η * gradient(X, Y, β)

        #FIND THE NUMBER OF FEATURES TO RETAIN
        Mi = int(k + (M - k)*max(0, (N_itr - 2*i)/(2*i*μ + N_itr)))
        #Mi = k + (M - k)*max(0, int((N_itr - 2*i)/(2*i*μ + N_itr)))
        β = feature_selection(β, Mi)
        indices = np.where(β == 0)

        #DELETE THE IRRELEVANT FEATURES FROM THE WEIGHTS AND DATA
        β = np.delete(β, indices)
        X = np.delete(X, indices, 1)
        validation = np.delete(validation, indices, 1)
        loss.append(lorenz_loss(X, Y, β))
    return β, X, validation, loss

#PREDICTION
def predict(X, Y, β):
    p = 1/(1 + np.exp(-np.dot(X, β)))
    p[p >= 0.5] = 1
    p[p < 0.5] = -1
    error = np.count_nonzero(Y - p)/Y.shape[0] * 100
    return error

#MAIN PROGRAM
f = open("report/results/results.txt", 'w')
l = open("report/results/losses.txt", 'w')

losses = []
for η, f_train_data, f_test_data, f_train_labels, f_test_labels in zip(η_master, training_data, test_data, training_labels, test_labels):
    print("Processing: " + f_train_data.split("/")[-2], file=f)
    print("Processing: " + f_train_data.split("/")[-2], file=l)
    for k in K:
        print("\tk = " + str(k), file=f)
        print("\tk = " + str(k), file=l)
        Xtrain = np.load(f_train_data)
        Ytrain = np.load(f_train_labels)
        Xtest = np.load(f_test_data)
        Ytest = np.load(f_test_labels)

        #TRAINING THE MODEL
        β, Xtrain, Xtest, loss = fsa(Xtrain, Ytrain, Xtest, η, k)
        if k == 10:
            losses.append(loss)

        #PREDICT NEW LABELS AND COMPUTE THE MISCLASSIFICATION ERROR
        error_train = predict(Xtrain, Ytrain, β)
        print("\t\tTraining error: " + str(error_train), file=f)
        error_test = predict(Xtest, Ytest, β)
        print("\t\tValidation error: " + str(error_test), file=f)

x_axis = range(1, 501)
fig, ax = plt.subplots()

ax.plot(x_axis, losses[0], label="Gisette")
ax.plot(x_axis, losses[1], label="Dexter")
ax.plot(x_axis, losses[2], label="Madelon")

legend = ax.legend(loc='best')
plt.title(r'Learning Rate eta = [5, 10, 0.01]')
plt.xticks([1, 100, 200, 300, 400, 500], [1, 100, 200, 300, 400, 500])
plt.xlabel("Iteration Number")
plt.ylabel("Training Loss")
plt.savefig("./report/figures/graph.eps", format="eps", dpi=1000, bbox_inches="tight")
```

```python
# Daniel Gonzalez, FSU Mathematics PhD
# Colton Piper, FSU Mathematics PhD
# Applied Machine Learning Assignment 6

import re
import numpy as np

training_data = ["./data/gisette/gisette_train.data", "./data/dexter/dexter_train.csv", "./data/madelon/madelon_train.data"]
test_data = ["./data/gisette/gisette_valid.data", "./data/dexter/dexter_valid.csv", "./data/madelon/madelon_valid.data"]

training_labels = ["./data/gisette/gisette_train.labels", "./data/dexter/dexter_train.labels", "./data/madelon/madelon_train.labels"]
test_labels = ["./data/gisette/gisette_valid.labels", "./data/dexter/dexter_valid.labels", "./data/madelon/madelon_valid.labels"]

#MAIN DATA PROCESSING
for f_train_data, f_test_data, f_train_labels, f_test_labels in zip(training_data, test_data, training_labels, test_labels):
    with open(f_train_data) as train, open(f_test_data) as test, open(f_train_labels) as train_labels, open(f_test_labels) as test_labels:
        print("Processing: " + str(f_train_data.split("/")[-1]))
        data = []
        valid = []
        data_labels = []
        valid_labels = []

        #INPUT DATA
        for line in train:
            data.append([float(x) for x in re.split(r'[, ]', line.strip().strip("\n"))])
        for line in test:
            valid.append([float(x) for x in re.split(r'[, ]', line.strip().strip("\n"))])
        for line in train_labels:
            data_labels.append([float(x) for x in re.split(r'[ ]', line.strip().strip("\n"))])
        for line in test_labels:
            valid_labels.append([float(x) for x in re.split(r'[ ]', line.strip().strip("\n"))])

        data = np.asarray(data)
        valid = np.asarray(valid)
        data_labels = np.asarray(data_labels)
        valid_labels = np.asarray(valid_labels)

        data_labels = data_labels.reshape(data_labels.shape[0])
        valid_labels = valid_labels.reshape(valid_labels.shape[0])

        #NORMALIZE
        avg = np.mean(data, axis=0)
        std = np.std(data, axis=0)

        data = (data - avg)/std
        data = data[:, (np.isfinite(data)).any(axis=0)]

        valid = (valid - avg)/std
        valid = valid[:, (np.isfinite(valid)).any(axis=0)]

        #RELABEL
        data_labels[data_labels != 1] = -1
        valid_labels[valid_labels != 1] = -1

        #OUTPUT DATA
        print("\t Writing normalized training data...")
        np.save("./data_norm/" + f_train_data.split("/")[-2] + "/" + f_train_data.split("/")[-1], data)

        print("\t Writing normalized validation data...")
        np.save("./data_norm/" + f_test_data.split("/")[-2] + "/" + f_test_data.split("/")[-1], valid)

        print("\t Writing training labels...")
        np.save("./data_norm/" + f_train_labels.split("/")[-2] + "/" + f_train_labels.split("/")[-1], data_labels)

        print("\t Writing validation labels...")
        np.save("./data_norm/" + f_test_labels.split("/")[-2] + "/" + f_test_labels.split("/")[-1], valid_labels)
```

"normalize.py" 66L, 2952C 1,