Project 2. Machine Learning (STA 5365). Aditya Ranjan Bhattacharya (arb17b) and Matthew Laird (mrl14b)

In Project 3, we made a simple Logistic Regression model as defined in the class lectures. We used a class LogisticRegression which consists of the various methods such as predict_proba(), predict(), fit(), gradient_ascent(), loss(), etc. Most of the methods are called by another method, such as fit() calls gradient_ascent(), and loss(), etc. We normalize the data as part of preprocessing, and then put a column of 1s in front of the feature set. This is done so that the equation.

$z = w_0 + \sum_{i=1}^{n} w_i * x_i$ becomes $z = \sum_{i=0}^{n} W_i * x_i$ for all samples, where x_0 for all samples is = 1. (Thus making w_0 the bias term).

The learning rate (η) for all the dataset is taken as 4 (apparently that works for all of the datasets pretty well, as evidenced by the dataset). The max_iter for the datasets are 1000 for Gisette and Madelon, and 3000 for Hill-Valley. For λ, we used the given value of 0.0001 The results for the experiments are given below.

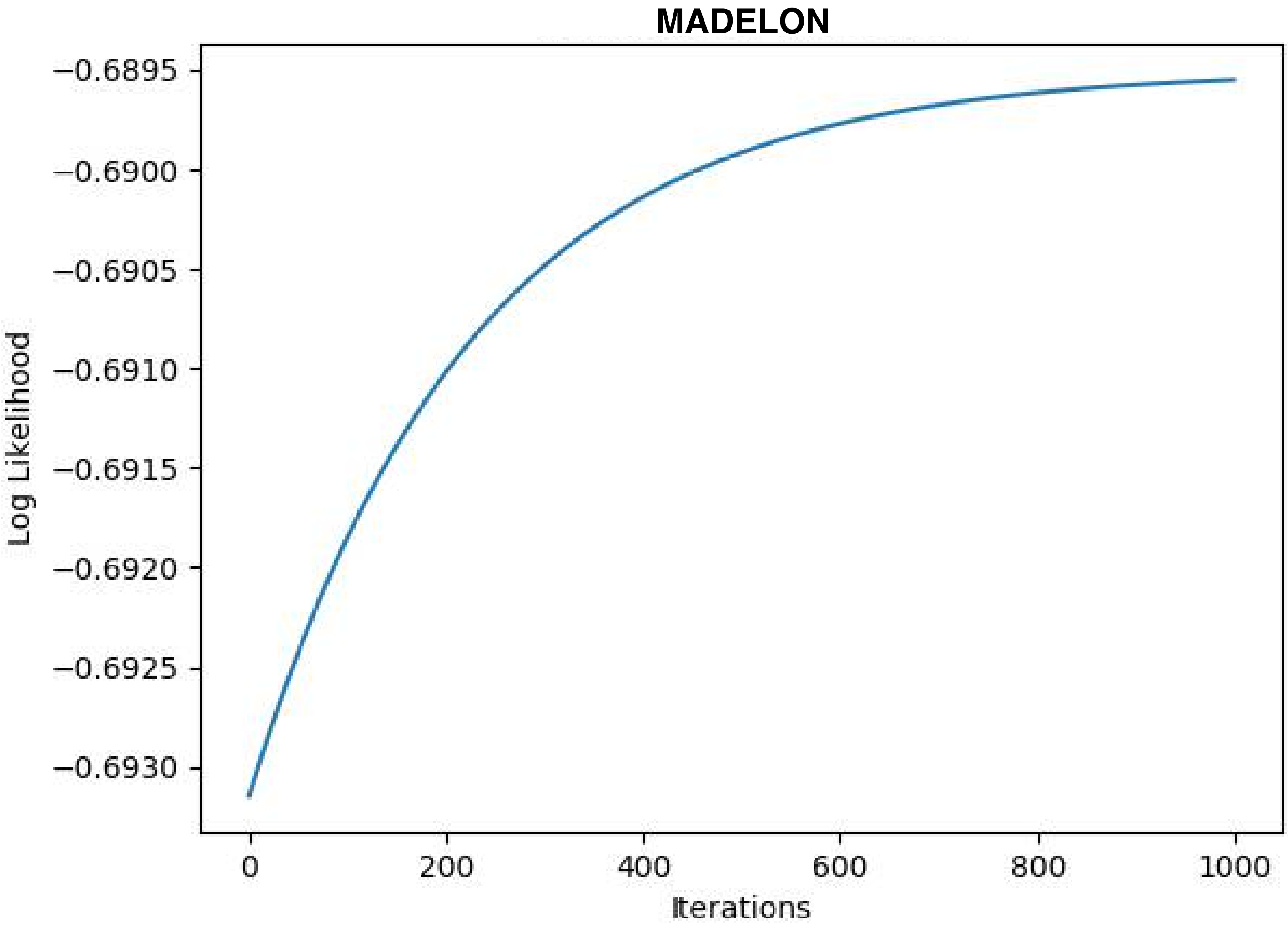
Test Mis-Classification Error of 2 variations of Logistic Regression on Testing Set for various Datasets

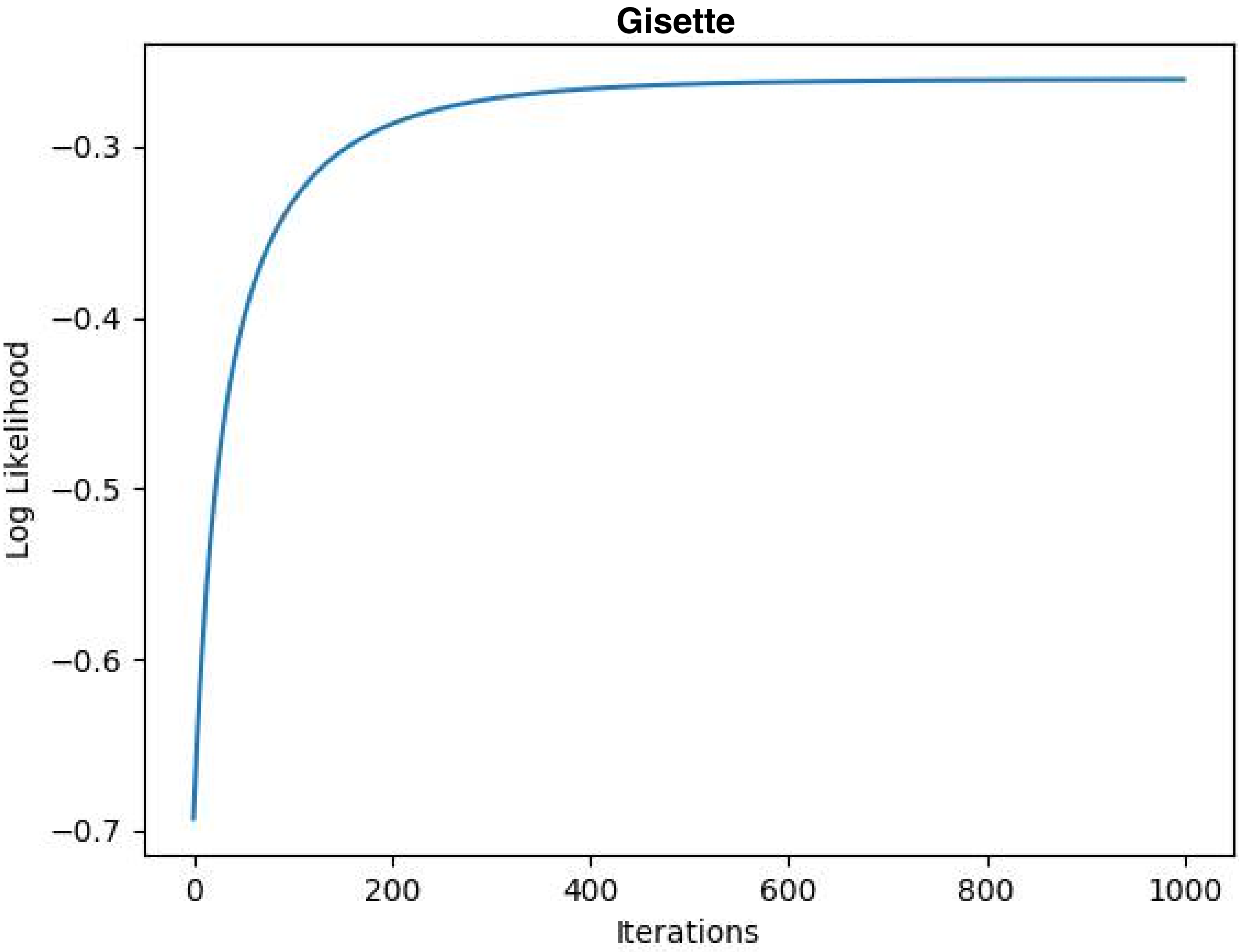
Dataset	Custom Implementation (Our Method)	State of the Art (sklearn)
Gisette	5.3	3.5
Hill-Valley	16.34	22.94
Madelon	40.84	41

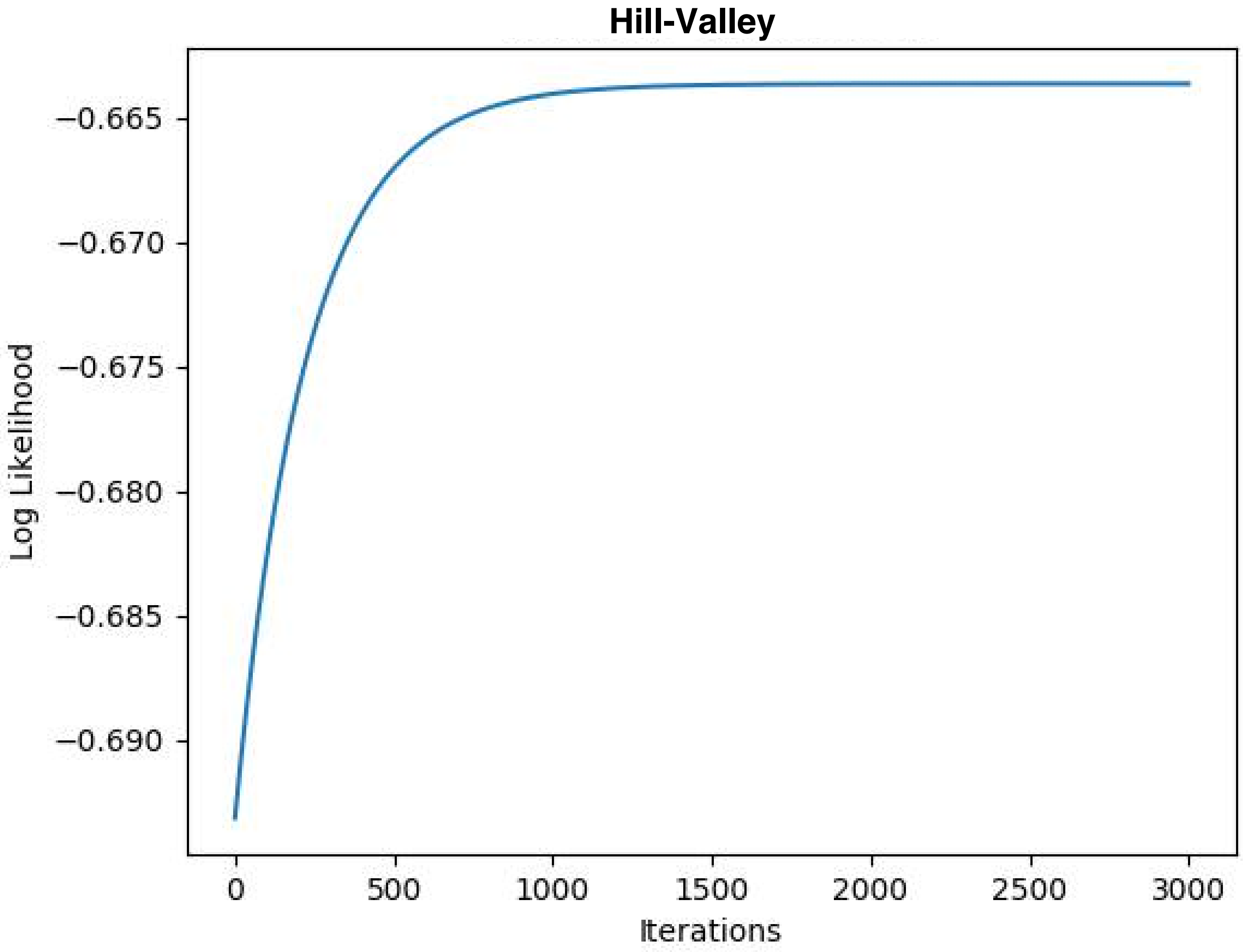
Test Mis-Classification Error of 2 variations of Logistic Regression on Training Set for various Datasets

Dataset	Custom Implementation (Our Method)	State of the Art (sklearn)
Gisette	6.8	3.94
Hill-Valley	16.01	20.3
Madelon	39.25	39.35

[1] Lars Buitinck et. al. API design for machine learning software: experiences from the scikit-learn project, ECML PKDD Workshop: Languages for Data Mining and Machine Learning, 2013, pages (108-122)


MADELON
Log Likelihood
−0.6895
−0.6900
−0.6905
−0.6910
−0.6915
−0.6920
−0.6925
−0.6930
0
200
400
600
800
1000
Iterations


Gisette
Log Likelihood
−0.3
−0.4
−0.5
−0.6
−0.7
0
200
400
600
800
1000
Iterations


Hill-Valley
Log Likelihood
−0.665
−0.670
−0.675
−0.680
−0.685
−0.690
0
500
1000
1500
2000
2500
3000
Iterations

```
import numpy

from sklearn import datasets
from sklearn import linear_model
from sklearn.preprocessing import normalize
import scipy
import sys
import matplotlib.pyplot as plt

def data_to_numpy(file):
                        #wilt is in .csv, hence we use numpy.genfromtxt which loads csv to a numpy array
faster and easily.
            X = numpy.genfromtxt(file)
            print(X)
            return X

#Labels Extracted From .label files.
def label_to_numpy(file):
    y = numpy.genfromtxt(file)
    print(y.shape)
    return y


def preprocess_labels(y):
            for i in range(0, y.shape[0]):
                        if(y[i] == -1):
                                    y[i] = 0
            return y

class LogisticRegression:

    def __init__(self, learning_rate=.001, lamda=0.001, max_iter=300):
        self.learning_rate = learning_rate
        self.lamda = lamda
        self.W = None
        self.max_iter = max_iter
        self.X_train = None
        self.log_likelihood = numpy.empty(0)

    def predict_proba(self, X_test):
            y_pred = []
            for i in range(0, X_test.shape[0]):
                        z = numpy.dot(self.W, X_test[i])
                        y_pred.append(scipy.special.expit(z))
            return y_pred

    def loss(self, y_pred, y_train):
            loss = []
            for i in range(0, y_train.shape[0]):
                        loss.append(y_train[i]*numpy.log(y_pred[i]) + (1-y_train[i])*numpy.log(1 - y_pred[i]))
            J = numpy.mean(loss)
            self.log_likelihood = numpy.append(self.log_likelihood, J)
            return J

    def gradient_ascent(self, X_train, y_train, y_pred):
            del_J = numpy.empty(0)
            for k in range(0, self.W.shape[0]):
                        gradient = numpy.dot(X_train[:,k],(y_train - y_pred))
                        del_J = numpy.append(del_J, (self.learning_rate*gradient/X_train.shape[0]))
            self.W = self.W - self.learning_rate*self.lamda*self.W + del_J
            return

    def fit(self, X_train, y_train):
            self.W = numpy.zeros(X_train.shape[1])
            for iterations in range(self.max_iter):
                        y_pred = self.predict_proba(X_train)
                        J = self.loss(y_pred, y_train)
                        print("Cost: ",J)
                        self.gradient_ascent(X_train, y_train, y_pred)
                        print(iterations)


    def predict(self, X_test):
            y = []
            y_pred = self.predict_proba(X_test)
            for i in range(0, X_test.shape[0]):
                        if(y_pred[i] > 0.5):
                                    y.append(0)
                        else:
                                    y.append(1)
            return y

    def scores(self, X_test, y_test):
            y = self.predict(X_test)
            mis_classification = 0
            for i in range(0, len(y)):
                        if(y[i] != y_test[i]):
                                    mis_classification += 1
            score = mis_classification/len(y)
            return score



X_train = data_to_numpy("../hill-valley/X.dat")
y_train = label_to_numpy("../hill-valley/Y.dat")

X_test = data_to_numpy("../hill-valley/Xtest.dat")
y_test = label_to_numpy("../hill-valley/Ytest.dat")

#X_train = data_to_numpy("../Gisette/gisette_train.data")
#y_train = label_to_numpy("../Gisette/gisette_train.labels")

#X_test = data_to_numpy("../Gisette/gisette_valid.data")
#y_test = label_to_numpy("../Gisette/gisette_valid.labels")




numpy.random.seed(0)
X_train = numpy.hstack((numpy.ones(X_train.shape[0])[:, numpy.newaxis], X_train))
X_test = numpy.hstack((numpy.ones(X_test.shape[0])[:, numpy.newaxis], X_test))

X_train = normalize(X_train)
X_test = normalize(X_test)

y_train = preprocess_labels(y_train)
y_test = preprocess_labels(y_test)


sklearn_model = linear_model.LogisticRegression()
sklearn_model.fit(X_train, y_train)

model = LogisticRegression(4, 0.001, 1000)
model.fit(X_train, y_train)

print("TESTING SET EFFICIENCY")
print("OURS: ",model.scores(X_test, y_test))
print("SKLEARN: ", sklearn_model.score(X_test, y_test))
print("TRAINING SET EFFICIENCY")
print("OURS: ",model.scores(X_train, y_train))
print("SKLEARN: ", sklearn_model.score(X_train, y_train))



plt.plot(model.log_likelihood)

plt.xlabel('Iterations')
plt.ylabel('Log Likelihood')

plt.title("Change of Log Likelihood w.r.t \n Iterations on MADELON")

plt.show()
```