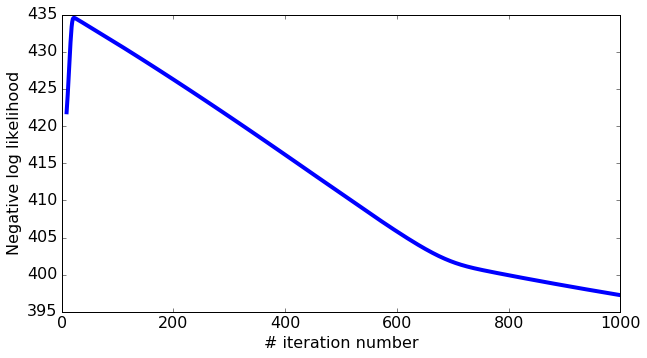
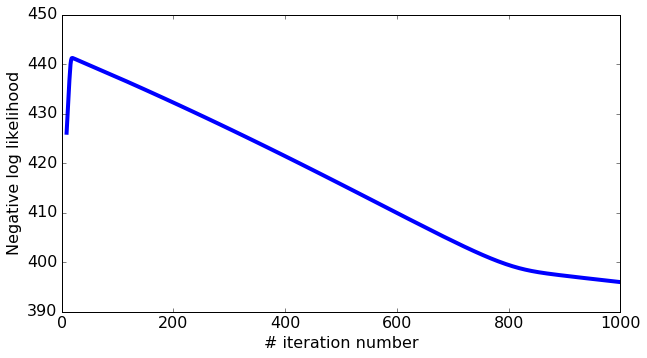
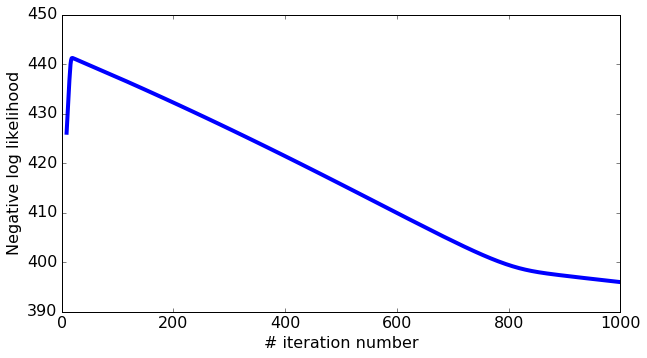
Hill-Valley data output:

1. The learning curves (plots of Negative log-likelihood vs # of iterations) were smoothed given the original plots were too noise to see the pattern.







The best learning rate ( i.e. step size ~ ). The curves don’t change much as the learning rate ranges from 4.4E-13 to 4.8E-13.

The L2 penalty coefficient values would not change the learning rate very much.

1. Error table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | L2 penalty | Learning rate | Training Error | Testing Error |
| Model\_1 | 0.1 |  | 0.8688298672012027 | 0.87103047073306239 |
| Model\_2 | 0.01 |  | 0.8684758097022649 | 0.87071454250324087 |
| Model\_3 | 0.001 |  | 0.86564879674920747 | 0.86863922085562084 |

Code in Python:

# -\*- coding: utf-8 -\*-

"""

Implement logistic regression to analyze Hill\_Valley data

"""

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

os.chdir("C:\\Users\\lancerts\\Dropbox\\statistics courses\\Applied Machine Learning\\HW\\HW3\\hill-valley")

#import data

X\_train = pd.read\_csv("X.dat",sep=" ",header=None).as\_matrix()

Y\_train= pd.read\_csv("Y.dat",sep=" ",header=None).as\_matrix().flatten()

X\_test= pd.read\_csv("Xtest.dat",sep=" ",header=None).as\_matrix()

Y\_test= pd.read\_csv("Ytest.dat",sep=" ",header=None).as\_matrix().flatten()

X\_train.shape

Y\_train.shape

#X\_train.plot.bar() #EDA of data

'''

produces probablistic estimate for P(y\_i = 1 | x\_i, w) estimate ranges between 0 and 1.. Feature\_matrix~ X, coefficients~ w

'''

def predict\_probability(feature\_matrix, coefficients):

# Take dot product of feature\_matrix and coefficients

score = np.dot(feature\_matrix,coefficients)

# Compute P(y\_i = +1 | x\_i, w) using the link function

predictions = 1/(1+np.exp(-score))

# return predictions

return predictions

"""

Here is the derivative of log likelihood:

errors ~ Yi-Yi\_hat

feature ~ Xi

coefficient ~ wi

l2\_penalty ~ lambda

"""

def feature\_derivative\_with\_L2(errors, feature, coefficient, l2\_penalty, feature\_is\_constant):

# Compute the dot product of errors and feature

derivative = np.dot(errors,feature)

# add L2 penalty term for any feature that isn't the intercept.

if not feature\_is\_constant:

## YOUR CODE HERE

derivative = derivative - 2\*l2\_penalty\*coefficient

return derivative

def compute\_log\_likelihood\_with\_L2(feature\_matrix,Y, coefficients, l2\_penalty):

indicator = (Y==+1)

scores = np.dot(feature\_matrix, coefficients)

lp = np.sum((indicator-1)\*scores - np.log(1. + np.exp(-scores))) - l2\_penalty\*np.sum(coefficients[1:]\*\*2)

return lp

"""

step\_size ~ learning rate 'eta'

l2\_penalty ~ coefficient 'lambda'

"""

def logistic\_regression\_with\_L2(feature\_matrix,Y, initial\_coefficients, step\_size, l2\_penalty, max\_iter):

coefficients = np.array(initial\_coefficients) # make sure it's a numpy array

neg\_log\_likelihood\_all = []

for itr in xrange(max\_iter):

# Predict P(y\_i = +1|x\_i,w) using your predict\_probability() function

## YOUR CODE HERE

predictions = predict\_probability(feature\_matrix, coefficients)

# Compute indicator value for (y\_i = +1)

indicator = (Y ==+1)

# Compute the errors as indicator - predictions

errors = indicator - predictions

for j in xrange(len(coefficients)): # loop over each coefficient

is\_intercept = (j == 0)

# Recall that feature\_matrix[:,j] is the feature column associated with coefficients[j].

# Compute the derivative for coefficients[j]. Save it in a variable called derivative

## YOUR CODE HERE

derivative = feature\_derivative\_with\_L2(errors, feature\_matrix[:,j], coefficients[j], l2\_penalty, is\_intercept)

# add the step size times the derivative to the current coefficient

## YOUR CODE HERE

coefficients[j] = coefficients[j] + step\_size\*derivative

# compute negative log-likelihood

lp = compute\_log\_likelihood\_with\_L2(feature\_matrix, Y, coefficients, l2\_penalty)

neg\_log\_likelihood\_all.append(-lp)

return coefficients, neg\_log\_likelihood\_all

#plot negative log-likelihood

def make\_plot(neg\_log\_likelihood\_all, smoothing\_window=10, label=''):

plt.rcParams.update({'figure.figsize': (9,5)})

neg\_log\_likelihood\_all\_ma = np.convolve(np.array(neg\_log\_likelihood\_all), \

np.ones((smoothing\_window,))/smoothing\_window, mode='valid')

plt.plot(np.array(range(smoothing\_window-1, len(neg\_log\_likelihood\_all))),

neg\_log\_likelihood\_all\_ma, linewidth=4.0, label=label)

plt.rcParams.update({'font.size': 16})

plt.tight\_layout()

plt.xlabel('# iteration number')

plt.ylabel('Negative log likelihood')

plt.legend(loc='lower right', prop={'size':14})

m = list(X\_train.shape)[1]

model\_1 = logistic\_regression\_with\_L2(X\_train, Y\_train, initial\_coefficients = np.zeros(m), step\_size = 4.2e-13, l2\_penalty = 0.1, max\_iter =1000)

make\_plot(list(model\_1)[1],)

model\_2 = logistic\_regression\_with\_L2(X\_train, Y\_train, initial\_coefficients = np.zeros(m), step\_size = 4.5e-13, l2\_penalty = 0.01, max\_iter =1000)

make\_plot(list(model\_2)[1],)

model\_3 = logistic\_regression\_with\_L2(X\_train, Y\_train, initial\_coefficients = np.zeros(m), step\_size = 4.8e-13, l2\_penalty = 0.001, max\_iter =1000)

make\_plot(list(model\_3)[1],)

"""

(b) Calculate the misclassification errors

"""

from sklearn.metrics import roc\_auc\_score

# model\_1

pred1\_train=predict\_probability(X\_train, list(model\_1)[0])

pred1\_test=predict\_probability(X\_test, list(model\_1)[0])

error\_train1=roc\_auc\_score(Y\_train, pred1\_train)

error\_test1=roc\_auc\_score(Y\_test, pred1\_test)

#model\_2

pred2\_train=predict\_probability(X\_train, list(model\_2)[0])

pred2\_test=predict\_probability(X\_test, list(model\_2)[0])

error\_train2=roc\_auc\_score(Y\_train, pred2\_train)

error\_test2=roc\_auc\_score(Y\_test, pred2\_test)

#model\_3

pred3\_train=predict\_probability(X\_train, list(model\_3)[0])

pred3\_test=predict\_probability(X\_test, list(model\_3)[0])

error\_train3=roc\_auc\_score(Y\_train, pred3\_train)

error\_test3=roc\_auc\_score(Y\_test, pred3\_test)