Thomas Freeman

3/31/22

Bioinformatics for CS

Report for Hw5

**Abstract:**

The goal of this assignment is to create multiple classifiers for the iris dataset given the following parameters. Firstly, our data set consists of only Versicolor and virginiana variants of flowers, two out of the three species that are available in the dataset. With this, we get our 100 initial inputs. Our first problem deals with performing the normal equation on the data set, also known as the closed form solution for linear regression. The second goal is to create a classifier that can show the sum of squared errors vs the number of iterations on the data. I used Epochs to describe iterations instead. Lastly, We classify the initial inputs using logistic regression. I used a CSV file to train the data.

**Method:**

For the first problem, We implement the normal equation over the dataset, getting our necessary weight values and plotting the line of regression as a result. The formula for the normal equation is as follows.

w = np.linalg.inv(X\_y.T.dot(X\_y)).dot(X\_y.T).dot(y)

Where our weight is equal to the transpose of the original matrix transposed times the dot product of the transposed matrix and the dot product of y. This results in giving two different values for w.

For the second problem, We build upon the linear regression even further by beginning the classify the data. The two species that we have are separated as one or negative one. This classifier has variables for the learning rate of our dataset and a given number of iterations, along with a random state to ensure that the results are the same every time. The results of running the classifier grant a graph of the sum squared error plotted against the epochs or number of iterations.

For our last problem, we simply run logistic regression on the remaining dataset. This results in giving a set of weight values for each of our inputs

**Discussion:**

For the three forms of classification that we performed in this assignment, here’s what I noticed. The linear regression with the closed form solution seemed to have the weight with the least amount of accuracy mainly because there wasn’t much classification being implemented for the first question. Just using the normal equation just performs linear regression on the dataset. Implementing the classifier on top of the linear regression would allow for more accuracy as all the data points could be categorized into two groups and not be operated on all at once. While logistic regression is meant to be more accurate in general, This model is generally linear rather than logistic. Implementing logistic regression proved to make very little difference on the dataset as manipulating it any further would take it out of linear, making the first two questions obsolete. This would explain why the weight of the logistic regression is so high, because its taking a set of inputs that are already linear out of context.

**Results:**

The results of the tests with the iris dataset in both linear and logistic regression can be seen within the appendix below.

**Appendix:**

The following code was implemented for questions one through three.

**Question 1**

In [ ]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

from sklearn import metrics

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

df = pd.read\_csv('Iris.csv')

df

Out[ ]:

|  | **Id** | **SepalLengthCm** | **SepalWidthCm** | **PetalLengthCm** | **PetalWidthCm** | **Species** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| **1** | 2 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| **2** | 3 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| **3** | 4 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| **4** | 5 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| **...** | ... | ... | ... | ... | ... | ... |
| **145** | 146 | 6.7 | 3.0 | 5.2 | 2.3 | Iris-virginica |
| **146** | 147 | 6.3 | 2.5 | 5.0 | 1.9 | Iris-virginica |
| **147** | 148 | 6.5 | 3.0 | 5.2 | 2.0 | Iris-virginica |
| **148** | 149 | 6.2 | 3.4 | 5.4 | 2.3 | Iris-virginica |
| **149** | 150 | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |

150 rows × 6 columns

In [ ]:

df.drop(df.index[df['Species'] == 'Iris-setosa'], inplace = True)

df.head()

Out[ ]:

|  | **Id** | **SepalLengthCm** | **SepalWidthCm** | **PetalLengthCm** | **PetalWidthCm** | **Species** |
| --- | --- | --- | --- | --- | --- | --- |
| **50** | 51 | 7.0 | 3.2 | 4.7 | 1.4 | Iris-versicolor |
| **51** | 52 | 6.4 | 3.2 | 4.5 | 1.5 | Iris-versicolor |
| **52** | 53 | 6.9 | 3.1 | 4.9 | 1.5 | Iris-versicolor |
| **53** | 54 | 5.5 | 2.3 | 4.0 | 1.3 | Iris-versicolor |
| **54** | 55 | 6.5 | 2.8 | 4.6 | 1.5 | Iris-versicolor |

In [ ]:

X = df['PetalLengthCm'] / 10

y = df['SepalLengthCm'] / 10

In [ ]:

X\_y = np.c\_[np.ones((100,1)), X]

# Normal equation is as follows

w = np.linalg.inv(X\_y.T.dot(X\_y)).dot(X\_y.T).dot(y)

print(w)

[0.29987107 0.66516293]

In [ ]:

X\_new = np.array([[0],[1]])

X\_new\_y = np.c\_[np.ones((2, 1)), X\_new]

y\_predict = X\_new\_y.dot(w)

y\_predict

Out[ ]:

array([0.29987107, 0.965034 ])

In [ ]:

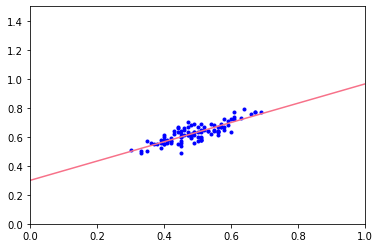
plt.axis([0,1,0,1.5])

plt.plot(X,y,"b.", y\_predict)

Out[ ]:

[<matplotlib.lines.Line2D at 0x7ff223d1ed10>,

<matplotlib.lines.Line2D at 0x7ff223d1ee90>]



**Question 2**

In [ ]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

In [ ]:

df = pd.read\_csv('Iris.csv')

df

Out[ ]:

|  | **Id** | **SepalLengthCm** | **SepalWidthCm** | **PetalLengthCm** | **PetalWidthCm** | **Species** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| **1** | 2 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| **2** | 3 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| **3** | 4 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| **4** | 5 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| **...** | ... | ... | ... | ... | ... | ... |
| **145** | 146 | 6.7 | 3.0 | 5.2 | 2.3 | Iris-virginica |
| **146** | 147 | 6.3 | 2.5 | 5.0 | 1.9 | Iris-virginica |
| **147** | 148 | 6.5 | 3.0 | 5.2 | 2.0 | Iris-virginica |
| **148** | 149 | 6.2 | 3.4 | 5.4 | 2.3 | Iris-virginica |
| **149** | 150 | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |

150 rows × 6 columns

In [ ]:

df.drop(df.index[df['Species'] == 'Iris-setosa'], inplace = True)

df.head()

Out[ ]:

|  | **Id** | **SepalLengthCm** | **SepalWidthCm** | **PetalLengthCm** | **PetalWidthCm** | **Species** |
| --- | --- | --- | --- | --- | --- | --- |
| **50** | 51 | 7.0 | 3.2 | 4.7 | 1.4 | Iris-versicolor |
| **51** | 52 | 6.4 | 3.2 | 4.5 | 1.5 | Iris-versicolor |
| **52** | 53 | 6.9 | 3.1 | 4.9 | 1.5 | Iris-versicolor |
| **53** | 54 | 5.5 | 2.3 | 4.0 | 1.3 | Iris-versicolor |
| **54** | 55 | 6.5 | 2.8 | 4.6 | 1.5 | Iris-versicolor |

In [ ]:

X = df[['PetalLengthCm', 'SepalLengthCm']].values

y = pd.factorize(df['Species'])[0]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.50)

print('#Training data points: {}'.format(X\_train.shape[0]))

print('#Testing data points: {}'.format(X\_test.shape[0]))

print('Species: {} (mapped from {}'.format(np.unique(y), np.unique(df['Species'])))

#Training data points: 50

#Testing data points: 50

Species: [0 1] (mapped from ['Iris-versicolor' 'Iris-virginica']

In [ ]:

sc = StandardScaler()

sc.fit(X\_train)

X\_train\_std = sc.transform(X\_train)

X\_test\_std = sc.transform(X\_test)

In [ ]:

class Classifier(object):

def \_\_init\_\_(self, eta=0.01, n\_iter=100, random\_state=1):

self.eta = eta # learning rate

self.n\_iter = n\_iter #

self.random\_state = random\_state

def fit(self, X, y):

rgen = np.random.RandomState(self.random\_state)

self.w\_ = rgen.normal(loc=0.0, scale=0.01, size=1+X.shape[1])

self.cost\_ = []

for i in range(self.n\_iter):

output = self.activation(X)

# Cost function

error = (y - output)

cost = (error\*\*2).sum() / 2.0

self.cost\_.append(cost)

# Update rule

self.w\_[1:] += self.eta \* X.T.dot(error)

self.w\_[0] += self.eta \* error.sum()

return self

def net\_input(self, X):

return np.dot(X, self.w\_[1:]) + self.w\_[0]

# calculates net input

def activation(self, X):

return self.net\_input(X)

def predict(self, X):

return np.where(self.activation(X) >= 0.0, 1, -1)

In [ ]:

classifier = Classifier(n\_iter=100, eta=0.01)

classifier.fit(X\_train\_std, y\_train)

# cost values

plt.plot(range(1, len(classifier.cost\_) + 1), classifier.cost\_, marker='o')

plt.xlabel('Epochs')

plt.ylabel('Sum-squared-error')

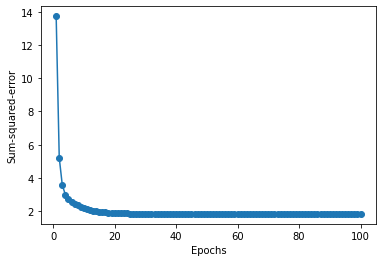
plt.show()

# testing accuracy

y\_pred = classifier.predict(X\_test\_std)

print('Misclassified samples: %d' % (y\_test != y\_pred).sum())

print('Accuracy: %.2f' % accuracy\_score(y\_test, y\_pred))



Misclassified samples: 28

Accuracy: 0.44

**Question 3**

In [ ]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

from sklearn import metrics

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split

df = pd.read\_csv('Iris.csv')

df

Out[ ]:

|  | **Id** | **SepalLengthCm** | **SepalWidthCm** | **PetalLengthCm** | **PetalWidthCm** | **Species** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| **1** | 2 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| **2** | 3 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| **3** | 4 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| **4** | 5 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| **...** | ... | ... | ... | ... | ... | ... |
| **145** | 146 | 6.7 | 3.0 | 5.2 | 2.3 | Iris-virginica |
| **146** | 147 | 6.3 | 2.5 | 5.0 | 1.9 | Iris-virginica |
| **147** | 148 | 6.5 | 3.0 | 5.2 | 2.0 | Iris-virginica |
| **148** | 149 | 6.2 | 3.4 | 5.4 | 2.3 | Iris-virginica |
| **149** | 150 | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |

150 rows × 6 columns

In [ ]:

df.drop(df.index[df['Species'] == 'Iris-setosa'], inplace = True)

df.head()

Out[ ]:

|  | **Id** | **SepalLengthCm** | **SepalWidthCm** | **PetalLengthCm** | **PetalWidthCm** | **Species** |
| --- | --- | --- | --- | --- | --- | --- |
| **50** | 51 | 7.0 | 3.2 | 4.7 | 1.4 | Iris-versicolor |
| **51** | 52 | 6.4 | 3.2 | 4.5 | 1.5 | Iris-versicolor |
| **52** | 53 | 6.9 | 3.1 | 4.9 | 1.5 | Iris-versicolor |
| **53** | 54 | 5.5 | 2.3 | 4.0 | 1.3 | Iris-versicolor |
| **54** | 55 | 6.5 | 2.8 | 4.6 | 1.5 | Iris-versicolor |

In [ ]:

df.describe

Out[ ]:

<bound method NDFrame.describe of Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm \

50 51 7.0 3.2 4.7 1.4

51 52 6.4 3.2 4.5 1.5

52 53 6.9 3.1 4.9 1.5

53 54 5.5 2.3 4.0 1.3

54 55 6.5 2.8 4.6 1.5

.. ... ... ... ... ...

145 146 6.7 3.0 5.2 2.3

146 147 6.3 2.5 5.0 1.9

147 148 6.5 3.0 5.2 2.0

148 149 6.2 3.4 5.4 2.3

149 150 5.9 3.0 5.1 1.8

Species

50 Iris-versicolor

51 Iris-versicolor

52 Iris-versicolor

53 Iris-versicolor

54 Iris-versicolor

.. ...

145 Iris-virginica

146 Iris-virginica

147 Iris-virginica

148 Iris-virginica

149 Iris-virginica

[100 rows x 6 columns]>

In [ ]:

df['Species'].value\_counts()

Out[ ]:

Iris-versicolor 50

Iris-virginica 50

Name: Species, dtype: int64

In [ ]:

fig = df[df.Species==''].plot(kind='scatter',x='SepalLengthCm',y='SepalWidthCm',color='orange', label='')

df[df.Species=='Iris-versicolor'].plot(kind='scatter',x='PetalLengthCm',y='SepalLengthCm',color='blue', label='versicolor',ax=fig)

df[df.Species=='Iris-virginica'].plot(kind='scatter',x='PetalLengthCm',y='SepalLengthCm',color='green', label='virginica', ax=fig)

fig.set\_xlabel("Petal Length")

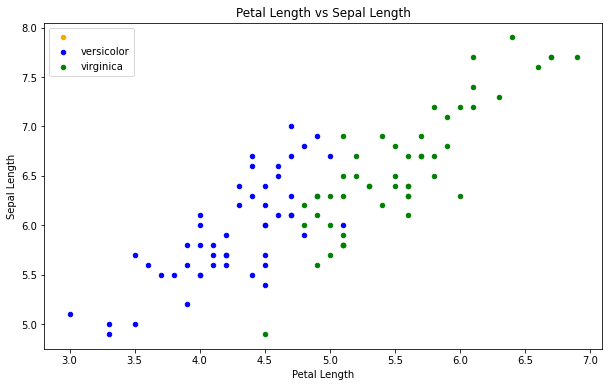
fig.set\_ylabel("Sepal Length")

fig.set\_title("Petal Length vs Sepal Length")

fig=plt.gcf()

fig.set\_size\_inches(10,6)

plt.show()



In [ ]:

train, test = train\_test\_split(df, test\_size = 0.3)# in this our main data is split into train and test

# the attribute test\_size=0.3 splits the data into 70% and 30% ratio. train=70% and test=30%

print(train.shape)

print(test.shape)

(70, 6)

(30, 6)

In [ ]:

train\_X = train[['SepalLengthCm','PetalLengthCm']]# taking the training data features

train\_y=train.Species# output of our training data

test\_X= test[['SepalLengthCm','PetalLengthCm']] # taking test data features

test\_y =test.Species #output value of test data

In [ ]:

model = LogisticRegression()

model.fit(train\_X,train\_y)

prediction=model.predict(test\_X)

print('The weight of Logistic regression is',metrics.accuracy\_score(prediction,test\_y))

The weight of Logistic regression is 0.9

In [ ]: