Step 1: Start a new Colab Notebook

Anthony Martinez

Assignment 4: Logistic Regression from Scratch

Feb 11 2022

Step 2: Write code to upload the Titanic data. Use the data file provided in Piazza

```
import numpy as np
import pandas as pd
import seaborn as sns
```

```
from google.colab import files
uploaded = files.upload()

import io
import pandas as pd

df = pd.read_csv(io.BytesIO(uploaded['titanic.csv']))
```

Choose Files titanic.csv

- **titanic.csv**(text/csv) - 106959 bytes, last modified: 2/2/2022 - 100% done

```
Saving titanic.csv to titanic (9).csv
```

Step 3: Let X be the pclass and y be survived.

```
X = df[['pclass']]
y = df.survived
```

	pclass	survived	name	sex	age	sibsp
0	1	1	Allen, Miss. Elisabeth Walton	female	29.0000	0
1	1	1	Allison, Master. Hudson Trevor	male	0.9167	1
2	1	0	Allison, Miss. Helen Loraine	female	2.0000	1

3	1	0	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1
4	1	0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1

Step 4: Divide into 80/20 train/test data sets.

```
# train/test split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

print('train size:', X_train.shape)
print('test size:', X_test.shape)
```

```
    train size: (1047, 1)
    test size: (262, 1)
```

5. Write a function to input a vector and return a vector of the sigmoid of those values. You can use np.log( ) and np.exp( ).

```
def sigmoid(vect):
  '''
  This function takes in a vector and returns a vector of the sigmoid of those values
  '''

  result_vect = np.zeros(shape=(len(vect)))


  # calculates the sigmoid of each element in the input vector and appends result to t
  for i in range (0,len(vect)):
    answer = 1 / (1 + np.exp(-vect[i]))
    result_vect[i] = answer

  return result_vect


#testing = [1,2,3,4]
#sigmoid(testing)
```

Step 6: Write code to calculate the coefficients on the training data using an iterative process.

a. set up a matrix where one column is the pclass and the other is all 1s

b. set up a small learning rate

c. set the weight matrix to any small values

d. function returns the weight and intercept

```
# a. set up a matrix where one column is the pclass and the other is all 1s

# create pclass list
pclass_col = X_train.pclass

# create ones_col list
ones_col = []
for i in range (1047):
  ones_col.append(1)

# use np.column_stack to set list as matrix columns
features_matrix = np.column_stack((pclass_col, ones_col))


#b. set up a small learning rate
learning_rate = .001


# c. set the weight matrix to any small values

weight_matrix = np.array([.01,.01])
```

```
(2,)
```

Step 7 Within the function, iterate through n steps (find the best n experimentally):

a. scores = dot product of features and weights (np.dot can be used)

b. predictions = sigmoid(scores)

c. error = target – predictions

d. gradient = dot product of features and error

```
# function for iterating through n steps

def model(features_matrix, weights_matrix):

  for i in range(200):


    # a. scores = dot product of features and weights
    scores = np.dot(features_matrix,weights_matrix)


    # b. predictions = sigmoid(scores)
    predictions = sigmoid(scores)



    # c. error = target - predictions
    error = np.subtract(y_train,predictions) #1047,


    # d. gradiant = dot product features and error

    transposed_features = np.transpose(features_matrix)

    gradiant = np.dot(transposed_features,error)  #error-1047, features 1047,2
    #print(gradiant.shape)


    # e. weight += learning_rate * gradient (experiment with different learning rates)
    weights_matrix += learning_rate * gradiant
    #print(weights_matrix)


  return weights_matrix
```

Step 8: Output coefficients b and w

```
answers = model(features_matrix,weight_matrix)
print(answers)
```

```
[-0.78718725  1.29103632]
```

Step 9: Run logistic regression in sklearn on the training data

```
from sklearn.linear_model import LogisticRegression

glm = LogisticRegression()
glm.fit(X_train, y_train)
```

```
LogisticRegression()
```

Step 10 Output the coefficients of the mode

```
print(glm.coef_, glm.intercept_)
```

```
[[-0.78602161]] [1.28948924]
```

Step 11: How similar are the coefficients? Write your analysis in a text cell

Commentary: My logistic regression algorithm got very similar coefficients to Sklearn’s algorithm. The coefficients of my Logistic Regression algorithm from scratch was [-0.78718725 1.29103632]. The Sklearn’s coefficients were [[-0.78602161]] [1.28948924].

✓ 0s completed at 7:07 PM ● ×