ASSGN_7_LR

April 7, 2022

[1]:

```
# MLFA ASSIGNMENT 7
# ROHIT RANJAN
# 20CS30066
```

[2]:

```
# importing required modules
import numpy as np
import pandas as pd
from tqdm import tqdm
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
```

[3]:

```
# reading the data csv
df = pd.read_csv('data.csv')
```

[4]:

```
# inserting a column with value 1 for vectorisation later
df.insert(0, 'Constant', 1)
```

[5]:

```
df = df[['Constant', 'age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg',
       'thalach', 'exang', 'oldpeak', 'num       ']]
```

[6]:

```
df.head(5)
```

[6]:

```
   Constant  age  sex  cp trestbps chol fbs restecg thalach exang  oldpeak  \
0         1   28    1   2      130  132   0       2     185     0      0.0
1         1   29    1   2      120  243   0       0     160     0      0.0
2         1   29    1   2      140    ?   0       0     170     0      0.0
3         1   30    0   1      170  237   0       1     170     0      0.0
4         1   31    0   2      100  219   0       1     150     0      0.0

   num
0            0
1            0
2            0
3            0
4            0
```

```
[7]: # removing rows with missing values
     df = df[df[df.columns]!='?']
     df = df.dropna()
     df = df.astype(float)
```

```
/home/tfjuror/anaconda3/lib/python3.7/site-
packages/pandas/core/ops/array_ops.py:253: FutureWarning: elementwise comparison
failed; returning scalar instead, but in the future will perform elementwise
comparison
  res_values = method(rvalues)
```

```
[8]: df.head(5)
```

```
[8]:    Constant   age  sex   cp  trestbps   chol  fbs  restecg  thalach  exang  \
     0       1.0  28.0  1.0  2.0     130.0  132.0  0.0      2.0    185.0    0.0
     1       1.0  29.0  1.0  2.0     120.0  243.0  0.0      0.0    160.0    0.0
     3       1.0  30.0  0.0  1.0     170.0  237.0  0.0      1.0    170.0    0.0
     4       1.0  31.0  0.0  2.0     100.0  219.0  0.0      1.0    150.0    0.0
     5       1.0  32.0  0.0  2.0     105.0  198.0  0.0      0.0    165.0    0.0

        oldpeak  num
     0      0.0         0.0
     1      0.0         0.0
     3      0.0         0.0
     4      0.0         0.0
     5      0.0         0.0
```

```
[9]: # all missing values have been dealt with
     np.sum(df.isna())
```

```
[9]: Constant      0
     age           0
     sex           0
     cp            0
     trestbps      0
     chol          0
     fbs           0
     restecg       0
     thalach       0
     exang         0
     oldpeak       0
     num           0
     dtype: int64
```

```
[10]: df.columns
```

```
[10]: Index(['Constant', 'age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg',
             'thalach', 'exang', 'oldpeak', 'num       '],
            dtype='object')
```

```
[11]: X_columns = ['Constant','age', 'sex', 'cp', 'trestbps', 'chol', 'fbs',␣
       ↪'restecg','thalach',
             'exang', 'oldpeak']
      Y_column = 'num       '
```

```
[12]: X = df[X_columns]
      Y = df[Y_column]

      print(X.shape)
      print(Y.shape)
```

```
(261, 11)
(261,)
```

```
[13]: # creating the train test split
      X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.
       ↪2,random_state=43)
```

```
[14]: print(X_train.shape)
      print(Y_train.shape)

      print(X_test.shape)
      print(Y_test.shape)
```

```
(208, 11)
(208,)
(53, 11)
(53,)
```

```
[15]: # weights randomly initialised
      W = np.random.rand(X_train.shape[1],1)
```

```
[16]: # initial weights
      print(W)
```

```
[[0.90541823]
 [0.83166119]
 [0.66342678]
 [0.49801271]
 [0.65829455]
 [0.60660356]
 [0.23746082]
 [0.534799  ]
 [0.10143478]
 [0.26997839]
 [0.21051869]]
```

[17]:

```
# hyperparameters defined here
epochs = 10000
lr = 1e-4
loss_array=[]
```

[18]:

```
# training loop with inbuilt updates via gradient descent
for epoch in tqdm(range(epochs),position=0, leave=True):
    if(epoch%2000 == 0):
        lr/=10
    L = 0
    del_L = np.zeros((X_train.shape[1],))
    for i in range(len(X_train)):
        row = np.array(X_train.iloc[i,:])
        row = np.reshape(row,(row.shape[0],1))
        H = (row.T.astype(float) @ W.astype(float))
        L += (H - float(Y_train.iloc[i]))**2
        for j in range(len(del_L)):
            del_L[j]+=(H - float(Y_train.iloc[i]))*row[j]
    L /= 2*len(X_train)
    del_L /= len(X_train)
    loss_array.append(L.item())
    for j in range(len(del_L)):
        W[j] = W[j] - lr* del_L[j]
```

```
100%|      | 10000/10000 [17:07<00:00,  9.73it/s]
```

[19]:

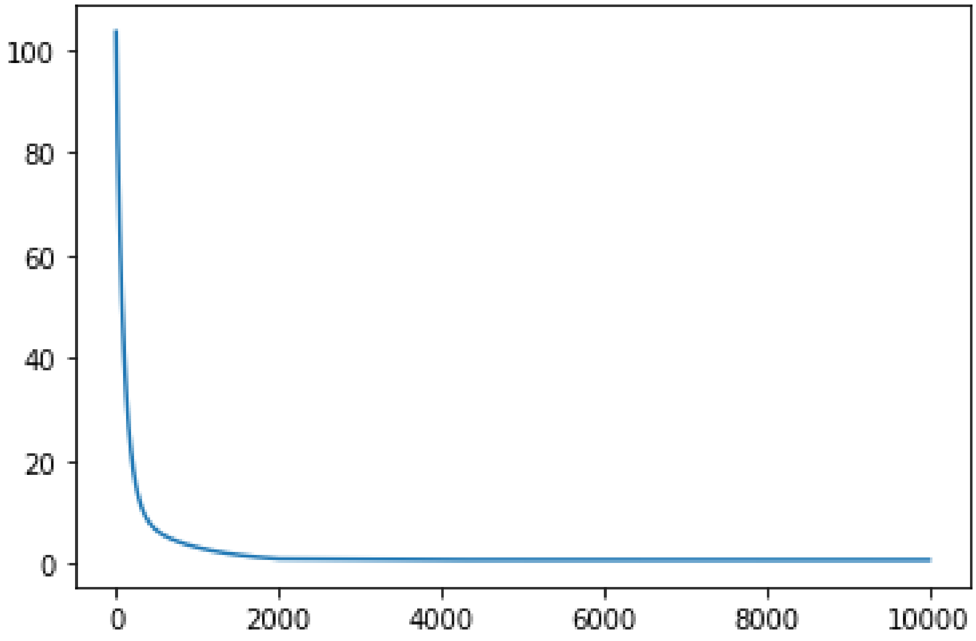
```
# finetuned weights after training
print(W)
```

```
[[ 0.89881455]
 [ 0.11374046]
 [ 0.65587431]
 [ 0.46341671]
 [-0.05247922]
 [-0.00368176]
 [ 0.2357119 ]
 [ 0.53042357]
 [-0.0023321 ]
 [ 0.26072079]
 [ 0.19678032]]
```

[20]:

```
# decreasing losses visualised
# first couple losses are removed from plotting to avoid skew due to random␣
 ↪initialisation
plt.plot(loss_array[2:])
```

```
[20]: [<matplotlib.lines.Line2D at 0x7f413d7baf50>]
```



```
[21]: # utility function to generate predictions
      def gen_preds(W,X):
          preds = []
          for i in range(len(X)):
              row = np.array(X.iloc[i,:])
              row = np.reshape(row,(row.shape[0],1))
              H = (row.T.astype(float) @ W.astype(float))
              threshed = 1 if H.item()>0.5 else 0
              preds.append(threshed)
          return np.array(preds)
```

```
[22]: preds_train = gen_preds(W,X_train)
      preds_test = gen_preds(W,X_test)
```

```
[23]: print('RMSE on train set',mean_squared_error(Y_train,np.array(preds_train)))
      print('RMSE on test set',mean_squared_error(Y_test,np.array(preds_test)))
```

```
RMSE on train set 0.3798076923076923
RMSE on test set 0.32075471698113206
```

```
[24]: print('Accuracy on train set',accuracy_score(Y_train,np.array(preds_train)))
      print('Accuracy on test set',accuracy_score(Y_test,np.array(preds_test)))
```

```
Accuracy on train set 0.6201923076923077
Accuracy on test set 0.6792452830188679
```

```
[ ]:
```