CO544 – Machine Learning and Data Mining

Lab 07

E/17/407

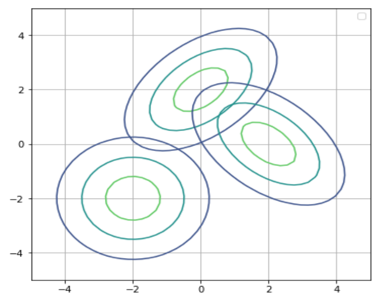
WIJESOORIYA H.D

Task 01

- **Lab 01**

Bivariate Gaussian Distribution

This is also called as bivariate normal distribution. In bivariate distributions we have 2 random variables. So in bivariate Gaussian distribution we have 2 random variables and both of them are normally distributed. In the following figure we can see the plots of 3 different bivariate gaussian distributions. The visualization of these distributions are 3D bell curves. But in 2D space we obtained contour lines as shown in the following figure.



Graph contains : 3 bivariate Gaussian distributions with following parameters.

m1=[0,2] c1=[[2,1] , [1,2]]

m2= [2,0] c2=[[2,-1],[-1,2]]

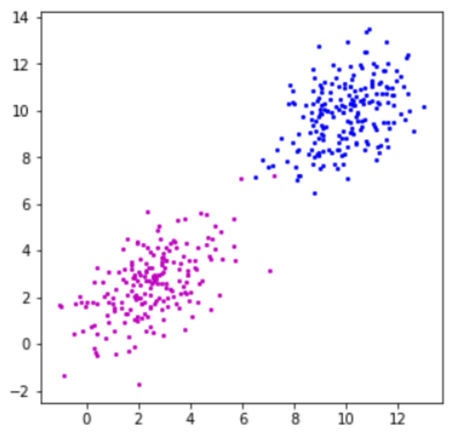
m3=[-2,-2] c3=[[2,0],[0,2]]

For the distribution with m1,c1 the mean value(maximum point) appeared at the top of the grid around the position (0,2). For the m2,c2 pair the mean value appeared around the position (2,0), and for the m3,c3 pair the mean value appeared around the position (-2,-2).

- **Lab 02**

Perceptron

In this part we implemented the perceptron algorithm. This is used in classification problems. It computes a linear classifier using a stochastic error correcting learning algorithm.



So first we generated 100 samples from 2 bivariate Gaussian densities and visualized it using a scatter plot and it is shown here. Here we can see two clusters and they are shown in magenta and blue colours. The perceptron algorithm will define the position of a vector which separate the two clusters.

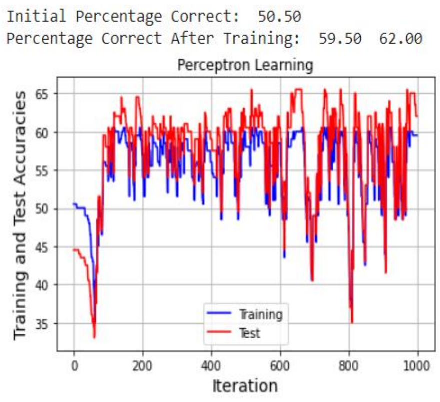
Values used in Gaussian distributions

m1 = [[2.5, 2.5]]

m2 = [[10.0, 10.0]]

C = [[2, 1], [1, 2]]

Then we obtained the accuracy of perceptron algorithm on test and train data sets. There we got 59.50% accuracy on training dataset and 62.00% accuracy on test dataset.



```
Initial Percentage Correct:  50.50
Percentage Correct After Training:  59.50  62.00
```

Finally, we did the same classification problem using in-build perceptron algorithm available in the scikitlearn package. There we obtained a high accuracy on both test and train data sets compared to the previous implementation.

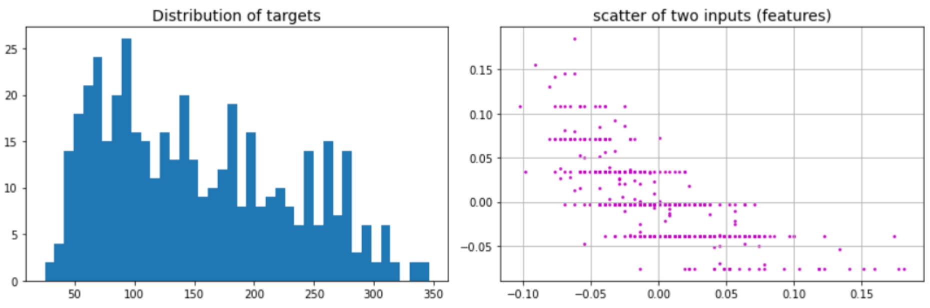
```
Accuracy on training set:   1.00
Accuracy on test set:   0.99
```

- **Lab 03**

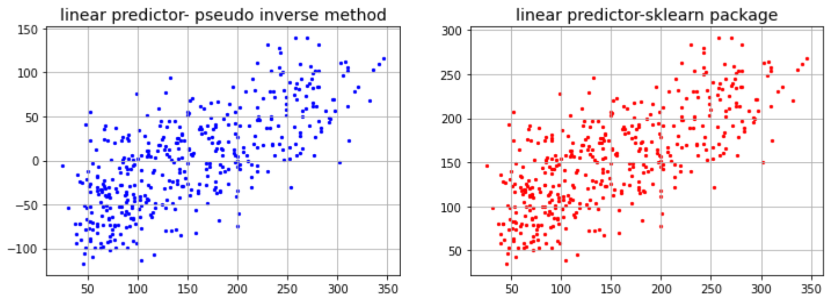
Linear Least Squares Regression

In this question, we worked with the Diabetes dataset from the UCI Machine Learning repository taken from the package sklearn.

- First, the data was loaded.
- Then the features and targets were inspected. (to get an idea about the targets a histogram was plotted, a scatter plot was plotted between two features to get an idea about the features)



- As shown in the above histogram the targets of the diabetes dataset are numerical values and they are labeled.
- So a linear regression model with a pseudo-inverse method has been developed to do the predictions.
- Finally, we did the same predictions using the linear model from sklearn and compared the results.

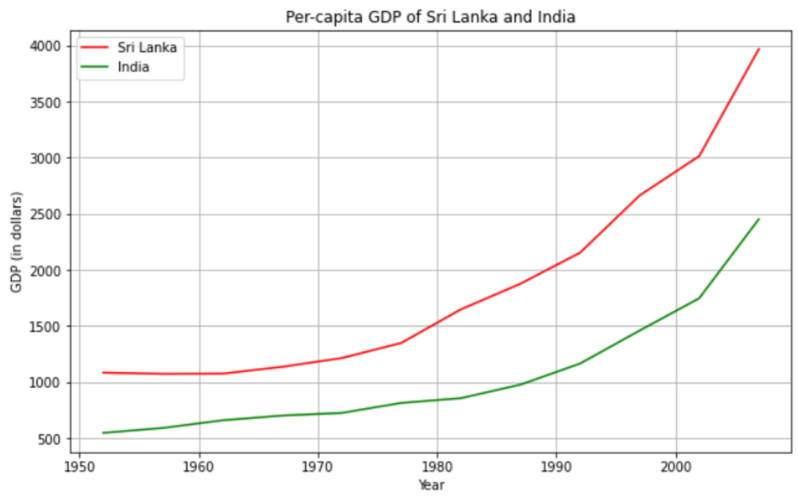


Even though we used the same linear regression technique in both methods we can see that the results are different.

- **Lab 04**

Line Plots

In this lab we learned about different kinds of plots that can be used in data visualization. Line plots can be created in Python with Matplotlib's pyplot library. This kind of plots show how a continuous variable changes over time. Here the continuous variable is represented on the y-axis, while the variable that measures time is plotted on the x-axis. In this lab we used a line plot to visualize the change in the per capita GDP of Sri Lanka and India over time. That graph is shown below.



According to this figure we can see that per capita GDP of Sri Lanka has increased faster than the per capita GDP of India from 1950-2000.

Task 03

Non-negative matrix factorization

This is a powerful dimensionality reduction approach. When we have all positive data and need to express it as a product of two matrices with all positive values, we can use this method. Compared to Principal Component Analysis, which preserves variance, restricting the factors to be positive increases representation sparsity. Therefore, NMF is regarded as a technique for achieving a parts-based representation.

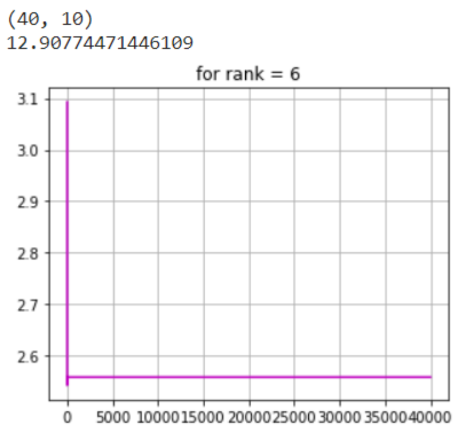
- First we completed the code – did the coding to update W and H

```
# Update W
AH_T = V@H.T
WHH_T =  W@H@H.T+ f[0]
for i in range(np.size(W, 0)):
  for j in range(np.size(W, 1)):
    W[i, j] = W[i, j] * AH_T[i, j] / WHH_T[i, j]
```

```
# Update H
W_TA = W.T@V
W_TWH = W.T@W@H+f[0]
for i in range(np.size(H, 0)):
  for j in range(np.size(H, 1)):
    H[i, j] = H[i, j] * W_TA[i, j] / W_TWH[i, j]
```

`

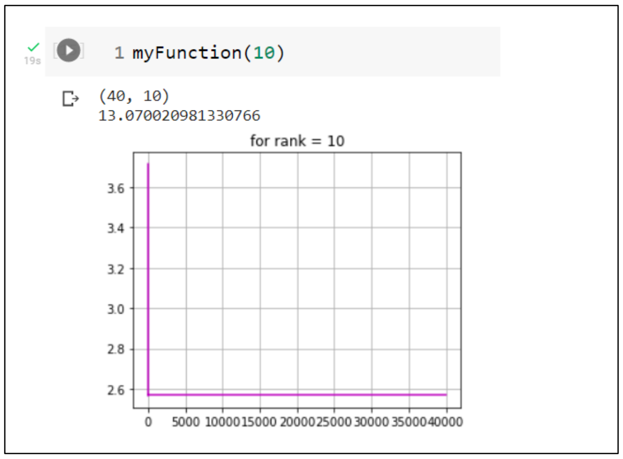
- When rank=6 we obtained the following result.



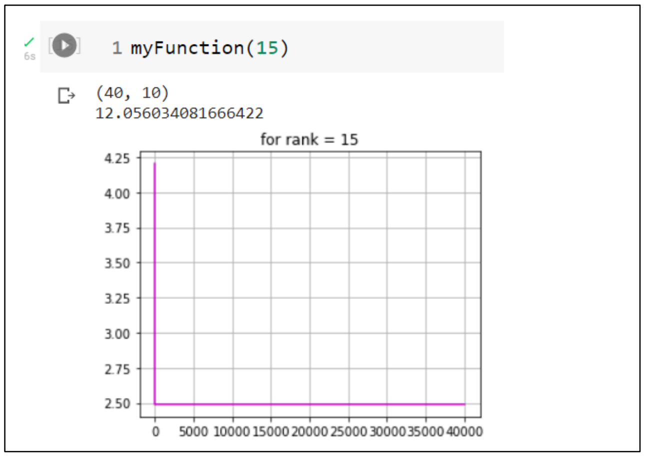
```
(40, 10)
12.90774471446109
```

As we are getting a 40x10 matrix as the output it is clear that the factorization is correct.

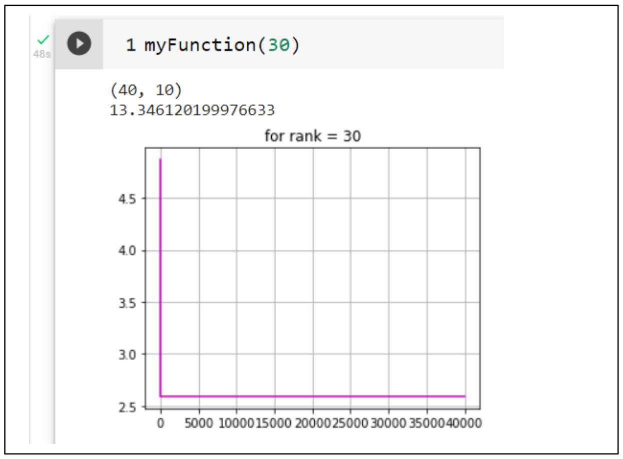
- Outputs that were obtained for the different ranks are given below.



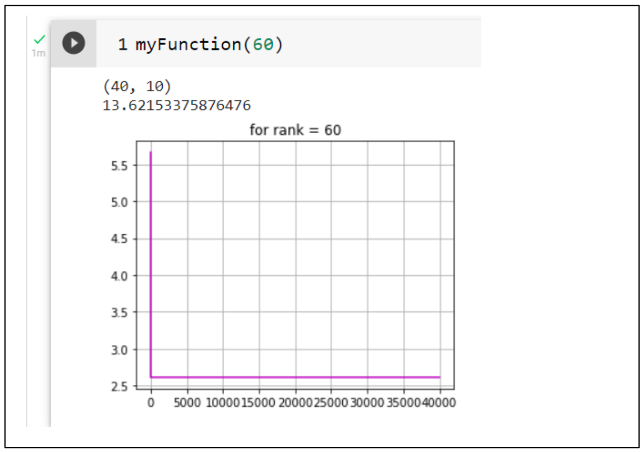
```
1 myFunction(10)
(40, 10)
13.070020981330766
```



```
1 myFunction(15)
(40, 10)
12.056034081666422
```



```
1 myFunction(30)
(40, 10)
13.346120199976633
```



```
1 myFunction(60)
(40, 10)
13.62153375876476
```