Machine Learning
Assignment 1 Report

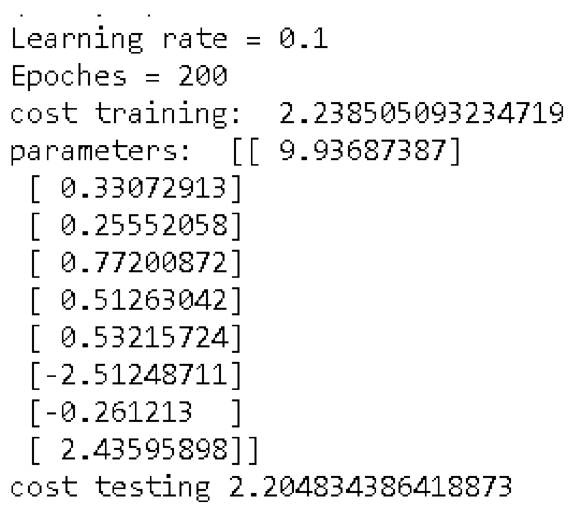
Utkarsh Dubey | 2019213

Question 1

Preprocessing

- Randomly shuffled the data
- Coded sex as
  - Infant = 0
  - Female = 1
  - Male = 2
- Introduced a column of 1s to account for the constant term in the equation
- Performed 8:2 train-test split
- Normalised the data by subtracting the mean and dividing it by the standard deviation to get lesser iterations for training.

Results Obtained

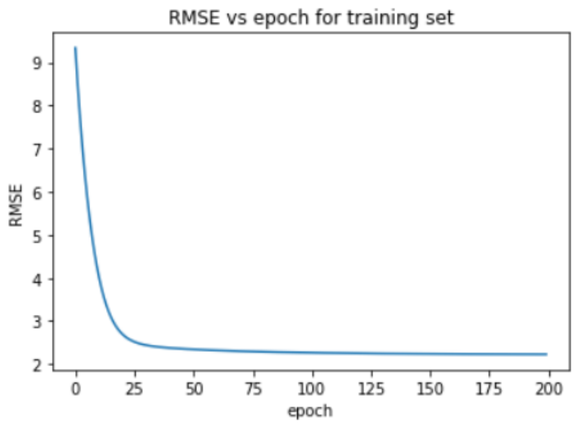


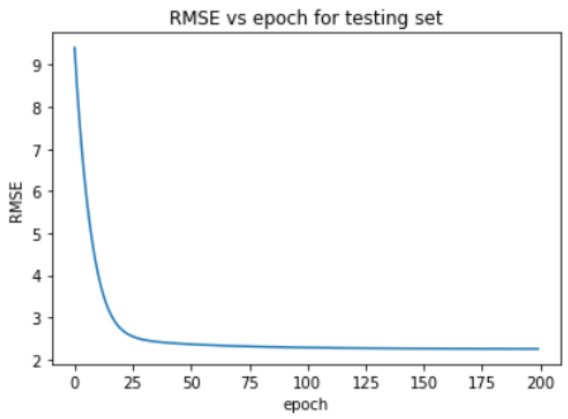
```
Learning rate = 0.1
Epoches = 200
cost training:  2.238505093234719
parameters:  [[ 9.93687387]
 [ 0.33072913]
 [ 0.25552058]
 [ 0.77200872]
 [ 0.51263042]
 [ 0.53215724]
 [-2.51248711]
 [-0.261213  ]
 [ 2.43595898]]
cost testing 2.204834386418873
```

These are the result of a learning rate of 0.1 and 200 iterations.

We get an RMSE of 2.23 for the training set and an RMSE of 2.20 for the testing set

Graphs -



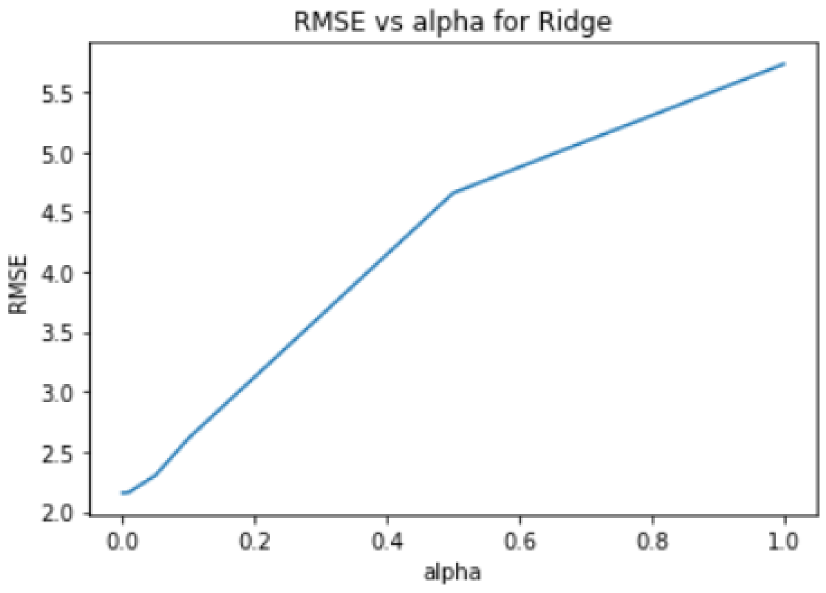


**While performing the ridge and lasso regression, I removed the normalisation which I did in preprocessing.*

Ridge Regression

Alphas used are - [1e-5,5e-5,1e-4,1e-3,5e-3,1e-2,5e-2,1e-1,5e-1,1]

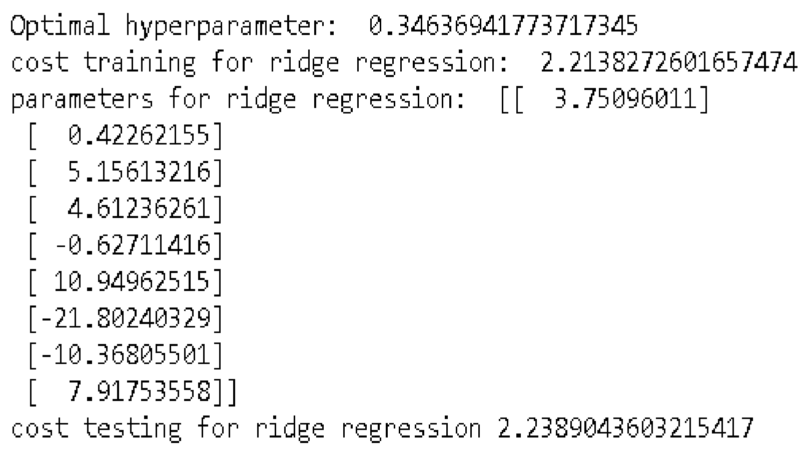
RMSE vs Alpha graph



Epoch - 1000

Learning rate - 0.1

Now, after using sklearn library we got the following results



```
Optimal hyperparameter:  0.34636941773717345
cost training for ridge regression:  2.2138272601657474
parameters for ridge regression:  [[  3.75096011]
 [  0.42262155]
 [  5.15613216]
 [  4.61236261]
 [ -0.62711416]
 [ 10.94962515]
 [-21.80240329]
 [-10.36805501]
 [  7.91753558]]
cost testing for ridge regression 2.2389043603215417
```

Learning rate - 0.1

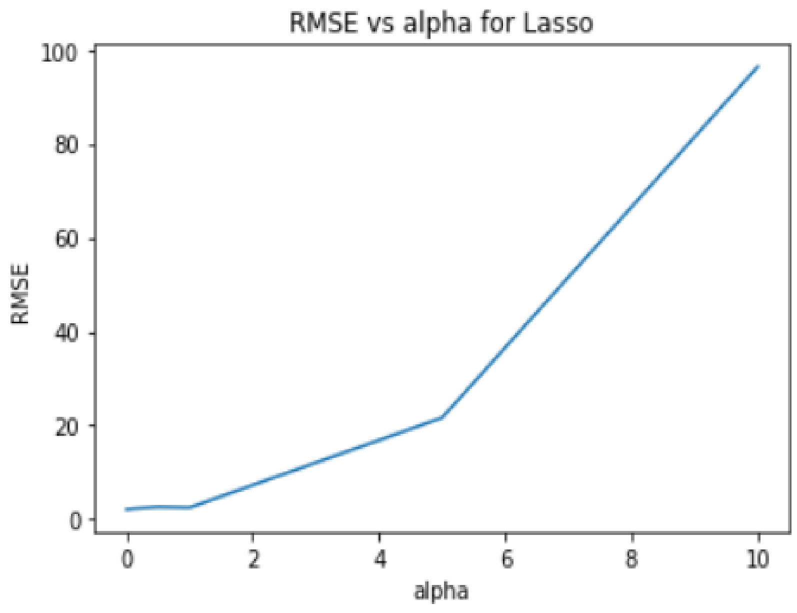
Epoch - 1000

We got the optimal hyperparameter as 0.34636941773717345

Lasso Regression

Alphas used are - [1e-5,5e-5,1e-4,1e-3,5e-3,1e-2,5e-2,1e-1,5e-1,1,5,10]

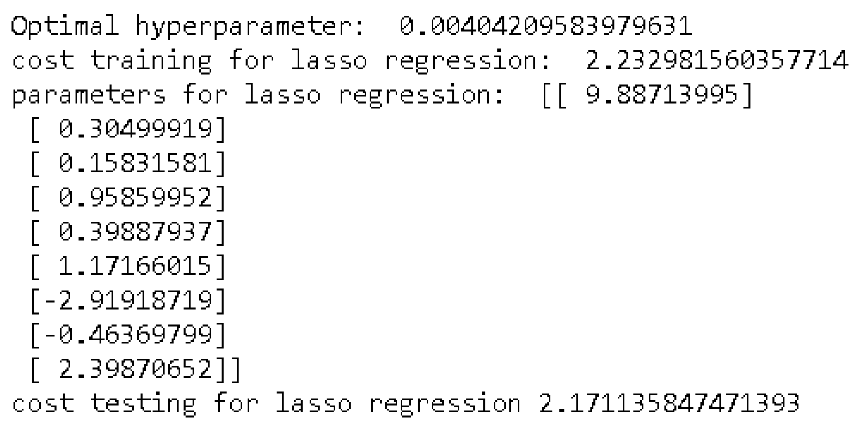
RMSE vs Alpha graph



Epoch - 1000

Learning rate - 0.1

Now after using the sklearn library we got the following results


```
Optimal hyperparameter:  0.00404209583979631
cost training for lasso regression:  2.232981560357714
parameters for lasso regression:  [[ 9.88713995]
 [ 0.30499919]
 [ 0.15831581]
 [ 0.95859952]
 [ 0.39887937]
 [ 1.17166015]
 [-2.91918719]
 [-0.46369799]
 [ 2.39870652]]
cost testing for lasso regression 2.171135847471393
```

Learning rate - 0.1

Epoch - 500

Optimal hyperparameter as 0.00404209583979631

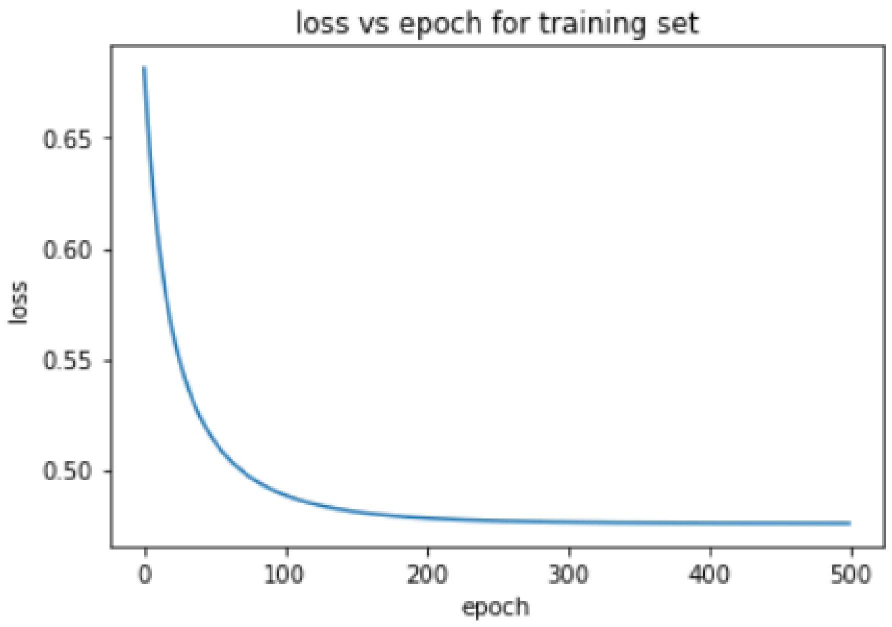
Question 2

Preprocessing

- Randomly shuffled the data
- Introduced a column of 1s to account for the constant term in the equation
- Performed 7:2:1 train-val-test split
- Normalised the data by subtracting the mean and dividing it by the standard deviation to get lesser iterations for training

Results Obtained

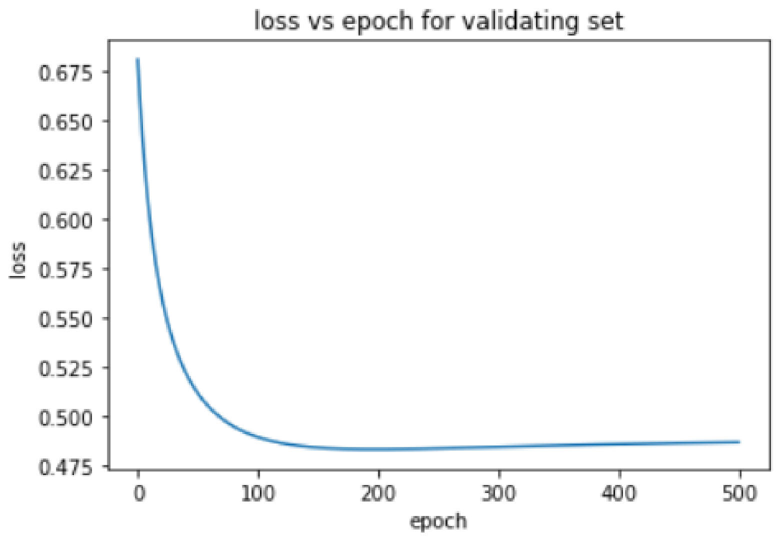
BGD Plots



Epoch - 500

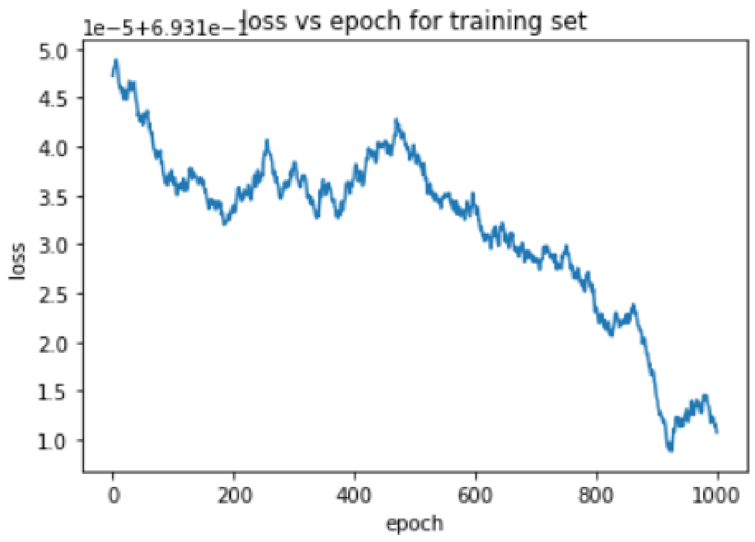
Learning rate - 0.1

Training loss is decreasing for training set



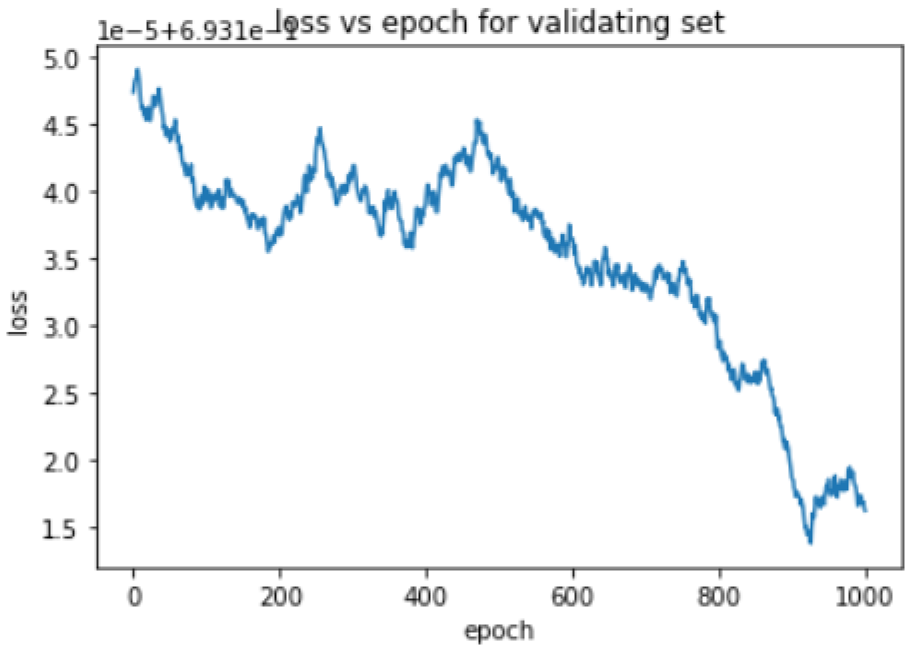
The loss curve on the validation set is seen to increase after certain epochs hence overfitting a little bit. Hence before that epoch, around 100, convergence occurs.

SGD Plots



Epoch - 1000

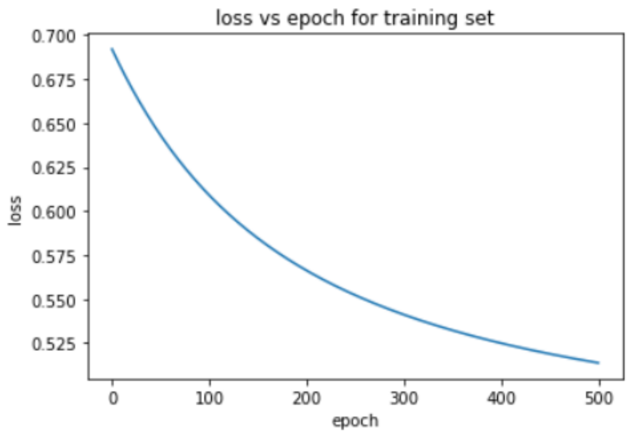
Learning rate - 0.001

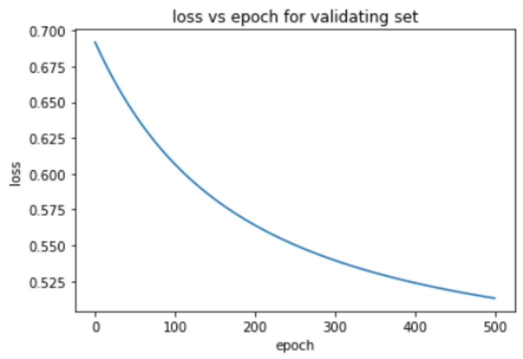


Running for different learning rates

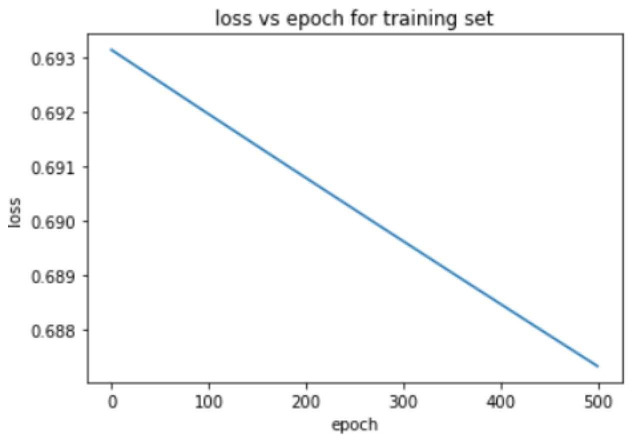
BGD

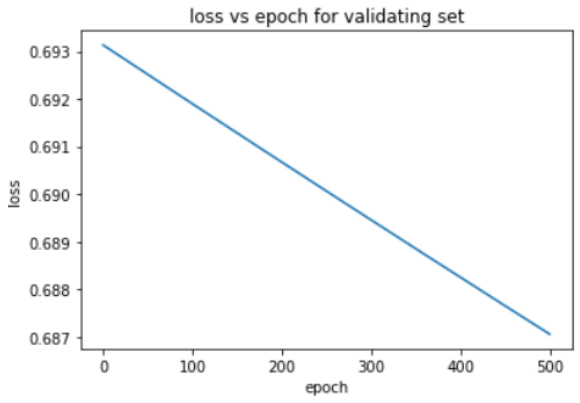
Learning rate - 0.01



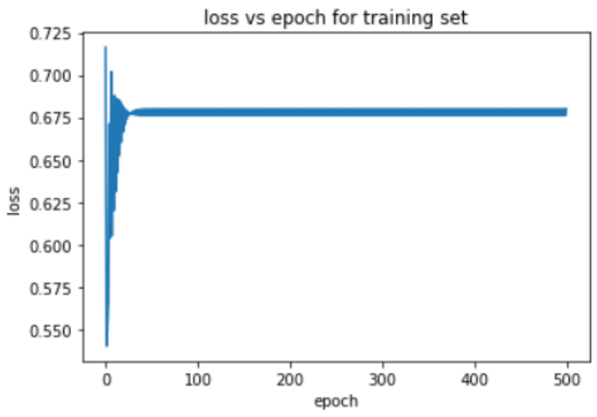


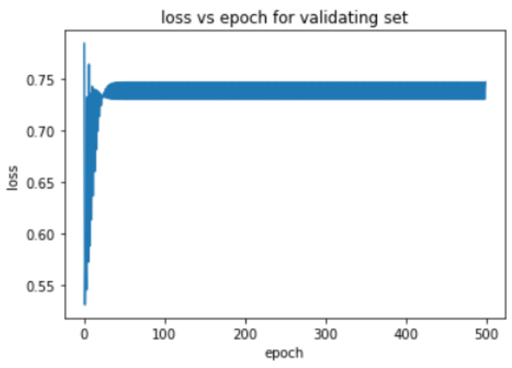
Learning rate - 0.0001





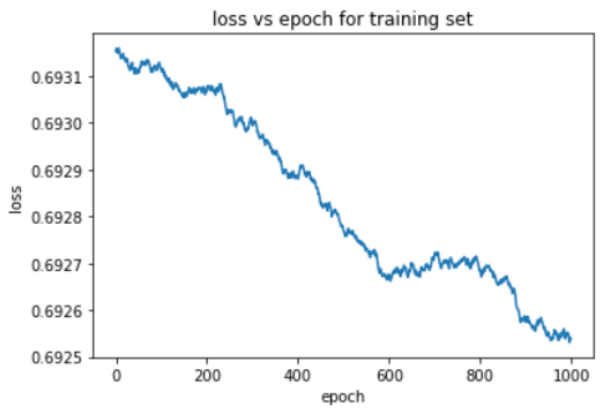
Learning rate - 10

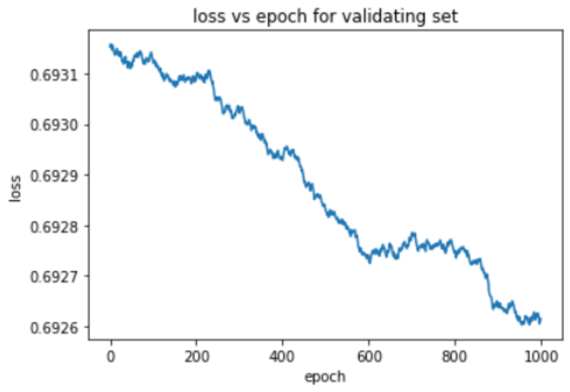




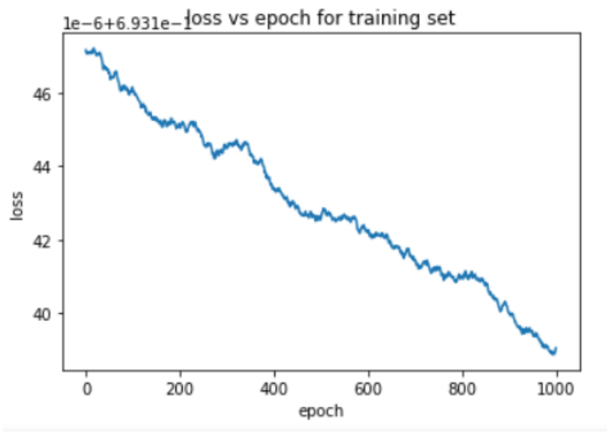
Comment on learning rates - For a learning rate of 0.01 we can see that convergence has begun to happen and if we increase the epochs to a little bit more, convergence will happen, For a learning rate of 0.0001 we almost see a straight line hence convergence is very very slow, Finally for the learning rate of 10 we see a graph which can't converge due to the high learning rate, as we miss the lowest point and start alternating between converging and diverging.

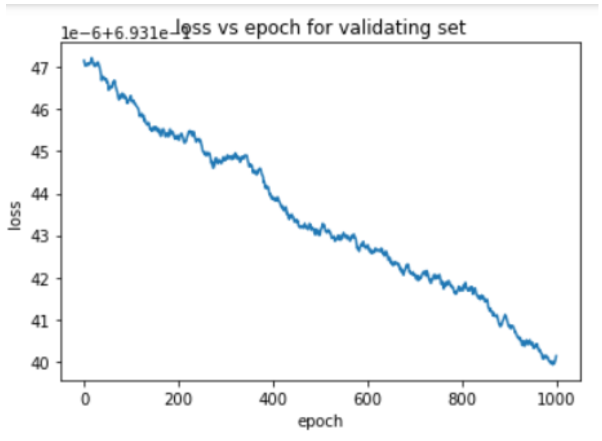
Learning Rate - 0.01



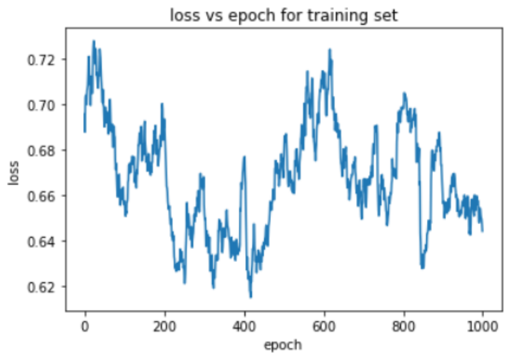


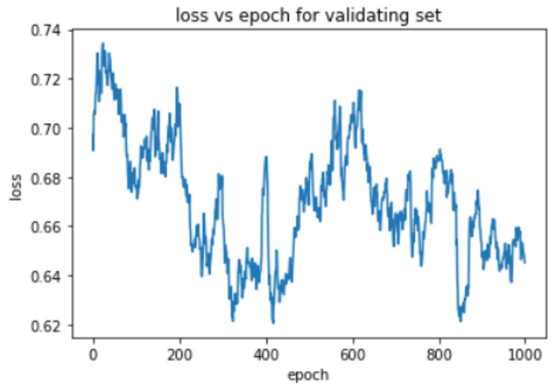
Learning Rate - 0.0001





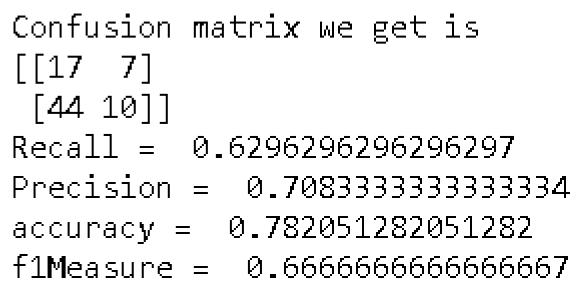
Learning rate - 10





Comment on learning rates - For a learning rate of 0.01 we can see that convergence has begun to happen and if we increase the epochs to a little bit more, convergence will happen, For a learning rate of 0.0001 we almost see a straight line hence convergence is very very slow, Finally for the learning rate of 10 we see a graph which can’t converge due to the high learning rate, as we miss the lowest point and start alternating between converging and diverging.

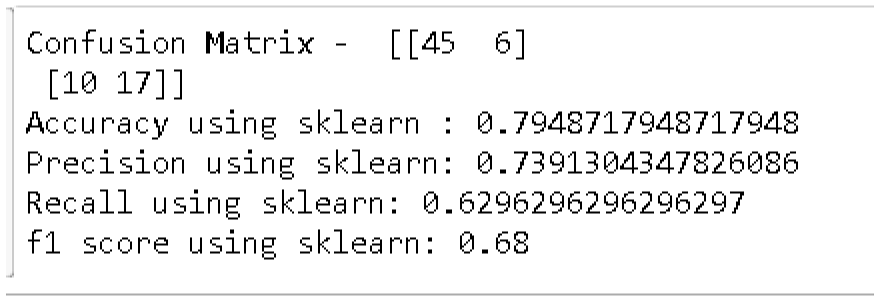
Reporting the confusion matrix



```
Confusion matrix we get is
[[17  7]
 [44 10]]
Recall =  0.6296296296296297
Precision =  0.7083333333333334
accuracy =  0.782051282051282
f1Measure =  0.6666666666666667
```

This confusion is made when considering BGD with a learning rate of 0.1 and epochs of 1000 via my implementation

Now using sklean library we get the following results -



```
Confusion Matrix -  [[45  6]
 [10 17]]
Accuracy using sklearn : 0.7948717948717948
Precision using sklearn: 0.7391304347826086
Recall using sklearn: 0.6296296296296297
f1 score using sklearn: 0.68
```

Max iteration - 25

My implementation converged at around 100 iterations and sklearn converged at about 25 iterations.

Hence we see that my model is forming good as we get almost near iterations for getting an accuracy of around 70 and sklearn is getting an accuracy of 78.

Question 3

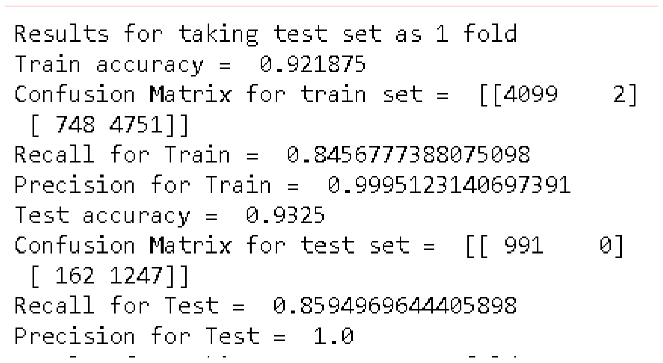
Preprocessing

- Randomly shuffled the data
- Binarized the pixel values to 0 and 255
- Deleted other classes which were not trousers and pullovers, was left with 12000 entries

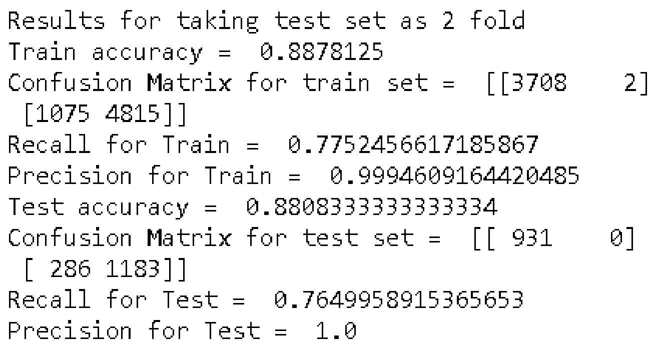
Choice of k for cross-validation -

I have chosen k as 5 because if I choose a high fold value the runtime will be very high. Choosing a low value of k will give me varying accuracy hence consistency will be affected and can result in overfitting. Overall taking 5 as my fold value.

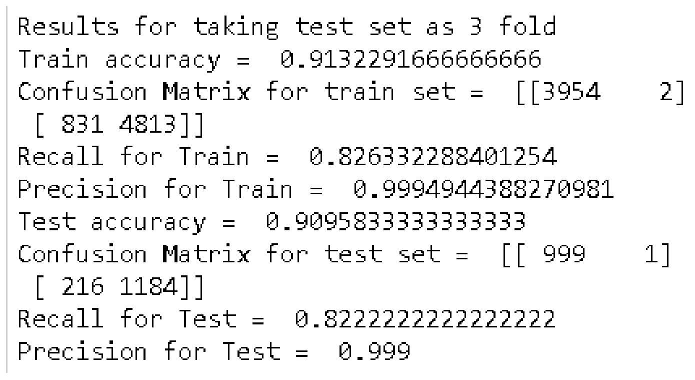
I got these results -



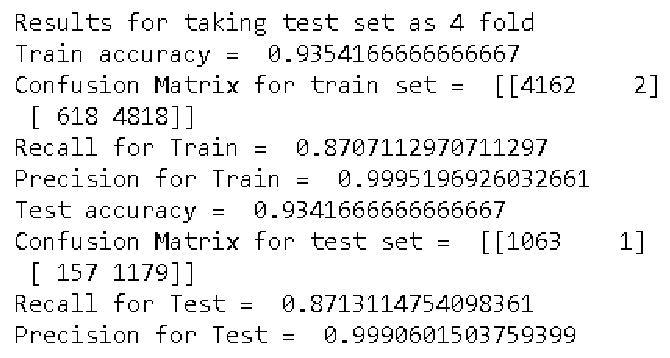
```
Results for taking test set as 1 fold
Train accuracy =  0.921875
Confusion Matrix for train set =  [[4099    2]
 [ 748 4751]]
Recall for Train =  0.8456777388075098
Precision for Train =  0.9995123140697391
Test accuracy =  0.9325
Confusion Matrix for test set =  [[ 991    0]
 [ 162 1247]]
Recall for Test =  0.8594969644405898
Precision for Test =  1.0
```



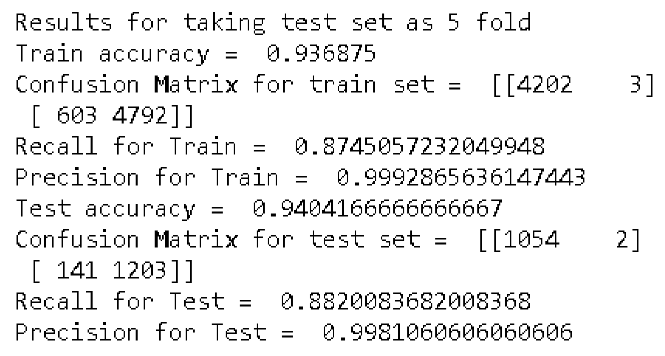
```
Results for taking test set as 2 fold
Train accuracy =  0.8878125
Confusion Matrix for train set =  [[3708    2]
 [1075 4815]]
Recall for Train =  0.7752456617185867
Precision for Train =  0.9994609164420485
Test accuracy =  0.8808333333333334
Confusion Matrix for test set =  [[ 931    0]
 [ 286 1183]]
Recall for Test =  0.7649958915365653
Precision for Test =  1.0
```



```
Results for taking test set as 3 fold
Train accuracy =  0.9132291666666666
Confusion Matrix for train set =  [[3954    2]
 [ 831 4813]]
Recall for Train =  0.826332288401254
Precision for Train =  0.9994944388270981
Test accuracy =  0.9095833333333333
Confusion Matrix for test set =  [[ 999    1]
 [ 216 1184]]
Recall for Test =  0.8222222222222222
Precision for Test =  0.999
```

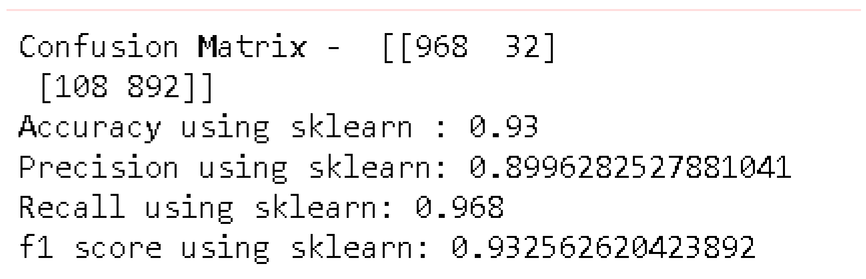


```
Results for taking test set as 4 fold
Train accuracy =  0.9354166666666667
Confusion Matrix for train set =  [[4162    2]
 [ 618 4818]]
Recall for Train =  0.8707112970711297
Precision for Train =  0.9995196926032661
Test accuracy =  0.9341666666666667
Confusion Matrix for test set =  [[1063    1]
 [ 157 1179]]
Recall for Test =  0.8713114754098361
Precision for Test =  0.9990601503759399
```



```
Results for taking test set as 5 fold
Train accuracy =  0.936875
Confusion Matrix for train set =  [[4202    3]
 [ 603 4792]]
Recall for Train =  0.8745057232049948
Precision for Train =  0.9992865636147443
Test accuracy =  0.9404166666666667
Confusion Matrix for test set =  [[1054    2]
 [ 141 1203]]
Recall for Test =  0.8820083682008368
Precision for Test =  0.9981060606060606
```

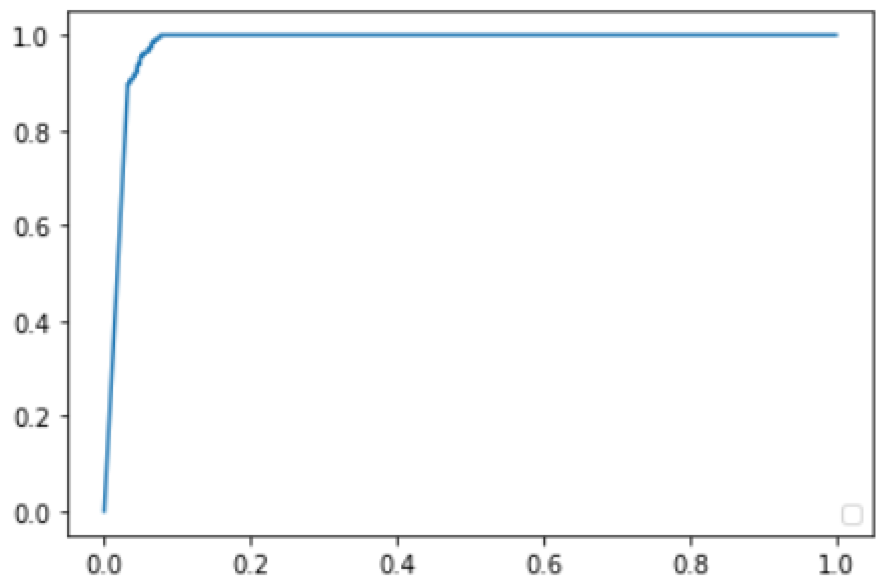
Now after using sklearn we got these results -



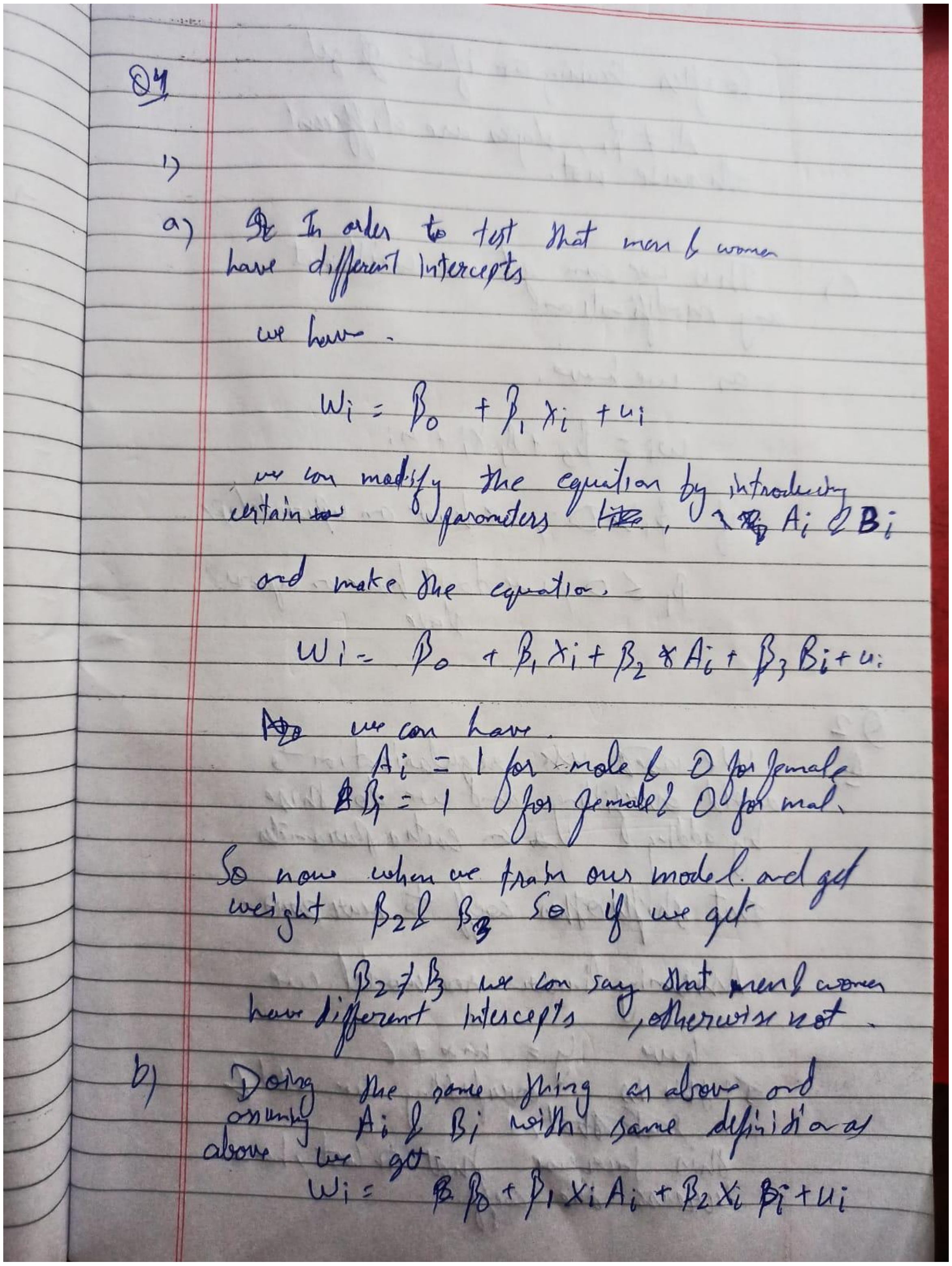
```
Confusion Matrix -  [[968  32]
 [108 892]]
Accuracy using sklearn : 0.93
Precision using sklearn: 0.8996282527881041
Recall using sklearn: 0.968
f1 score using sklearn: 0.932562620423892
```

We can observe a 93% accuracy, whereas my implementation with 5 fold cv gave the best accuracy of about 94%. Hence my implementation is doing great.

ROC curve -



Question 4



Q4

1)

a) ~~[illegible]~~ In order to test that men & women have different intercepts

we have.

$$W_i = \beta_0 + \beta_1 x_i + u_i$$

we can modify the equation by introducing certain ~~[illegible]~~ parameters ~~like~~, ~~[illegible]~~ A_i & B_i

and make the equation.

$$W_i = \beta_0 + \beta_1 x_i + \beta_2 \cancel{x} A_i + \beta_3 B_i + u_i$$

~~As~~ we can have.

$A_i = 1$ for male & 0 for female

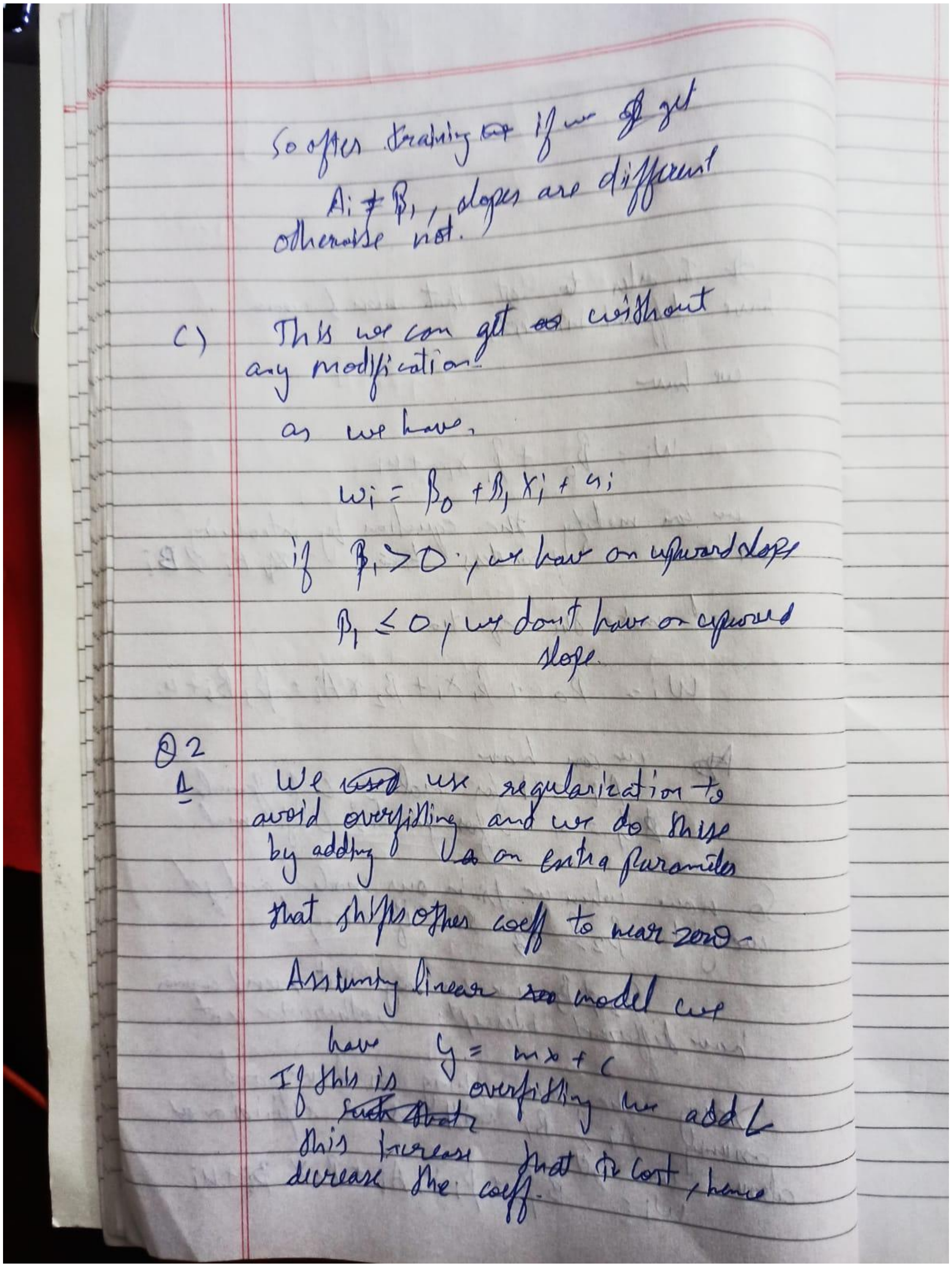
~~B~~ $B_i = 1$ for female & 0 for male.

So now when we train our model and get weight β_2 & β_3 so if we get

$\beta_2 \neq \beta_3$ we can say that men & women have different intercepts, otherwise not.

b) Doing the same thing as above and assuming A_i & B_i with same definition as above we got.

$$W_i = \cancel{\beta} \beta_0 + \beta_1 X_i A_i + \beta_2 X_i B_i + u_i$$



So after training ~~as~~ if we ~~g~~ get
$A_i \neq \beta_1$, slopes are different
otherwise not.

c) This we can get ~~as~~ without any modification.

as we have,

$$w_i = \beta_0 + \beta_1 X_i + u_i$$

if $\beta_1 > 0$, we have an upward slope

$\beta_1 \leq 0$, we don't have an upward slope.

Q2

A We ~~used~~ use regularization to avoid overfitting and we do this by adding ~~as~~ an extra parameter that shifts other coeff to near zero.

Assuming linear ~~reg~~ model we have $y = mx + c$

If this is ~~such that~~ overfitting we add L
this increase that ~~the~~ cost, hence decrease the coeff.