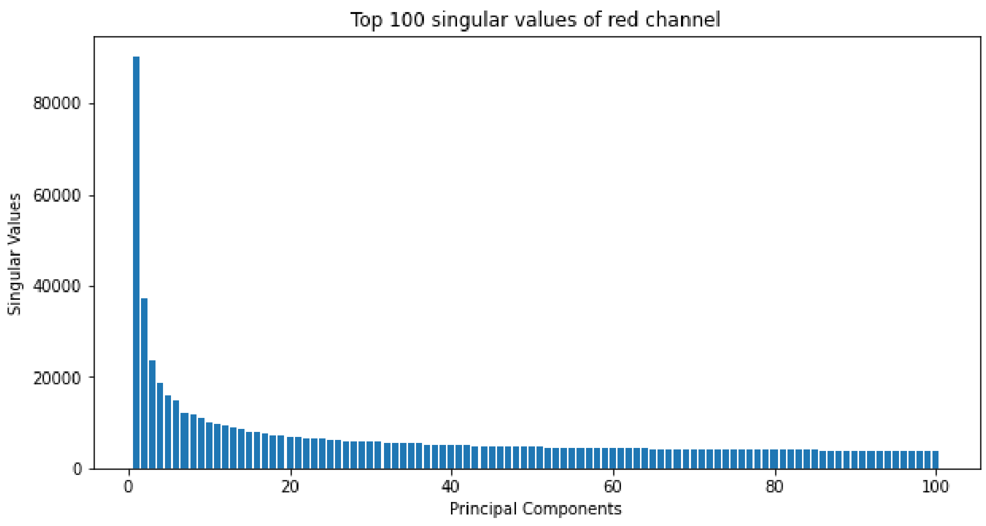
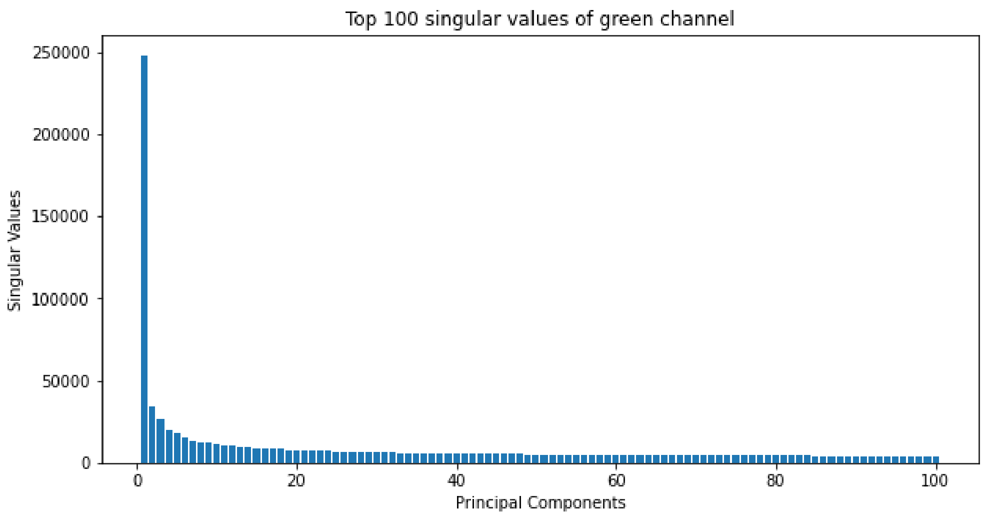
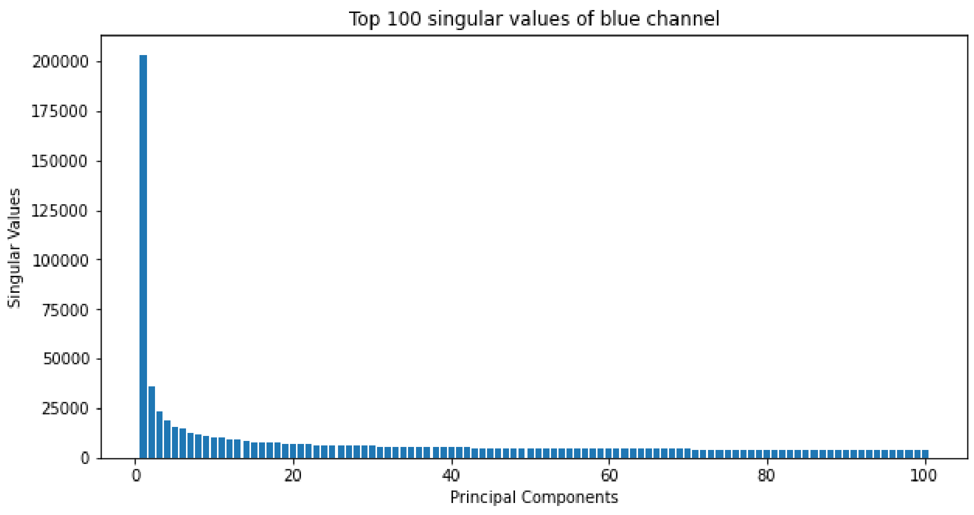
CS-464-HW2-Report

Arda Türkoğlu

Q1.1







Q1.1 Red channel proportion of variance explained

```
[47.11393802676904,
 7.952398158799591,
3.187992684940145,
2.013399832186432,
1.4477222540535275,
1.284663752439323,
 0.8466659651701824,
0.7908505224205905,
 0.6976683277465813,
0.5806389755233324]
```

Q1.1 Green channel proportion of variance explained

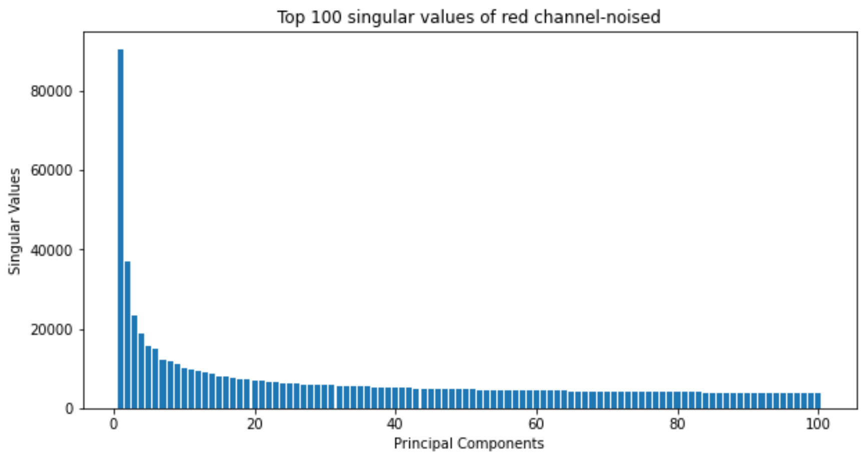
```
[86.12758303705608,
 1.6685031258736691,
 0.9896501868043278,
0.5531449460645225,
0.4308439550149289,
0.3229555791134812,
0.22772251706226543,
0.21421407200275105,
0.20122826496089943,
 0.16431715003134828]
```

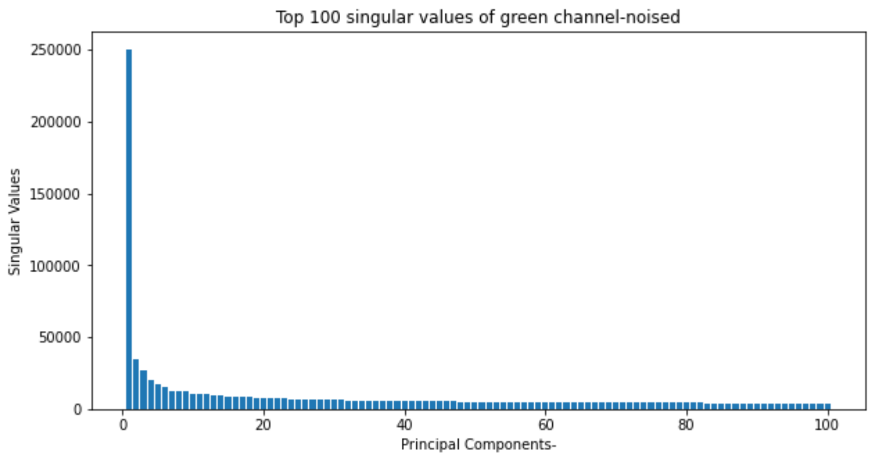
Q1.1 Blue channel proportion of variance explained

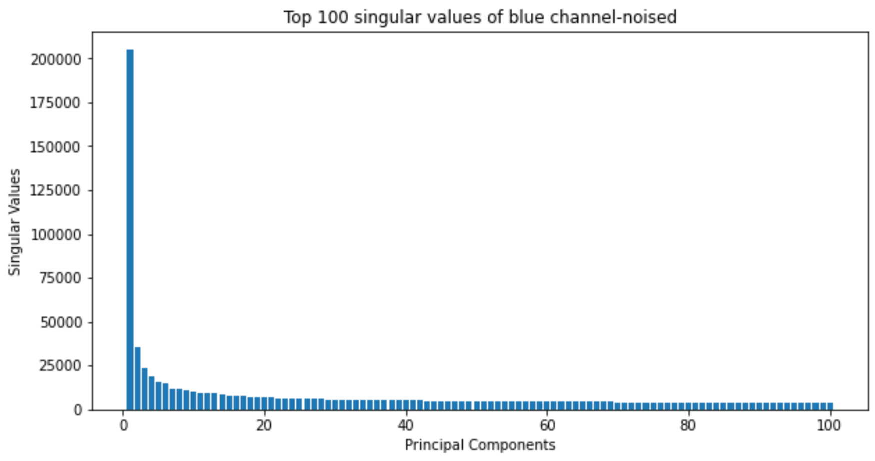
```
[82.0702520434635,
2.5296966504527996,
1.0761551121053032,
0.6943988243256742,
 0.4962333630619571,
0.4450383685185839,
0.2913060039899343,
 0.27118356858956644,
0.23834556215748615,
 0.19892552435429234]
```

As a discussion, we can see that green channel has the most impact on principal components and singular values. First component of each color channel has the biggest singular value. Singular values show square roots of the eigenvalues of the corresponding components. Green channel’s first component has the biggest eigen value. Therefore, our principal components lay on mostly green channel. That components defines the characteristics (important features) of the dataset. PCA tries to get maximum possible variance of the components. So, as we can see from the variance explained values of each channel, first component has the most variance among other components and first 10 component explains the 90% of the total variance explained.

Q1.2







Q1.2 Noised Image-Red channel proportion of variance explained

[47.25366890221805,

7.935784708399816,

3.1799148541973863,

2.007145718436636,

1.443474249400563,

1.280891823824321,

0.844111318302976,

0.788485406327923,

0.6956070138940526,

0.5788871622670431]

Q1.2 Noised Image-Green channel proportion of variance explained

[86.3428091370545,

1.642439632875729,

0.9742894941301582,

0.5444463239867909,

0.424060168382487,

0.31787019177667863,

0.22418366511406254,

0.21087142013029,

0.19803808441196716,

0.16172749308031764]

Q1.2 Noised Image-Blue channel proportion of variance explained

[82.31829895458127,

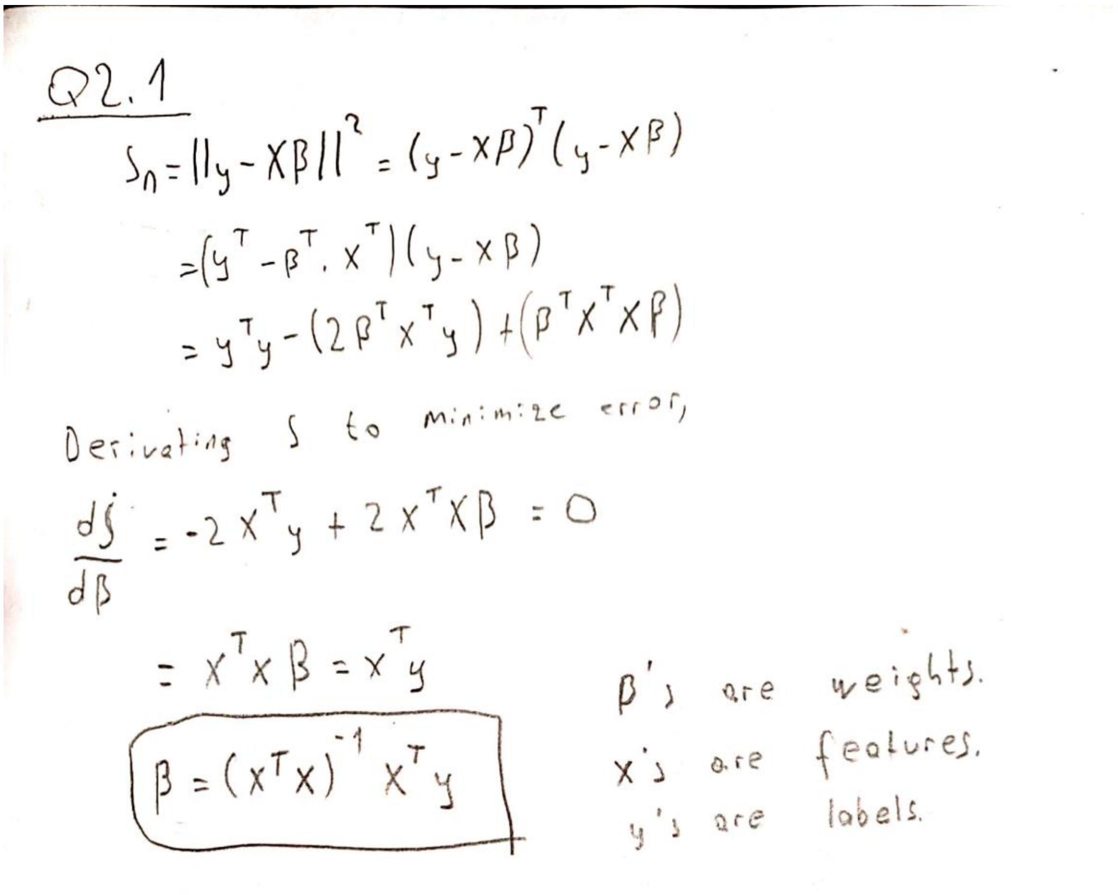
2.4941116086823314,

1.0610912540798159,

```
0.6845870760645609,
0.48930757397961977,
0.43884184241485735,
 0.28722035930564244,
0.267383627736829,
0.2350057259084704,
0.19613143033723107]
```

For the second part of the question we add Gaussian noise N(mean,square of the variance) with the factor 0.01. Since noise factor is too small, our singular values and principal components almost did not change. If we give a noise with bigger magnitude, noise will change the color channels. Therefore, components and singular values of the channel may increase and our plot gives increased result per principal component. We can reconstruct the image from the principal components by multiplying the component values with the transpose of the eigenvalues and adding mean to them. Each new principal component means new reconstructed image. Each component increases total variance explained. For example, out green channel holds 86% of the variance but if we add some other components that variance explained increased to 90 -95% of the variance. So, number of the principal components increases the accuracy of the reconstructed image. More principal components mean more correctly reconstructed images. We can decrease the presence of the artificially added noise on the images reconstructed by increasing the principal component that we used to reconstruct the image. Also, we added noise with means and square root of the variances. So, if we subtract the means and variances from the reconstructed image, we can get rid of noise.

Q2.1



Q2.1

$$S_n = ||y - X\beta||^2 = (y - X\beta)^T (y - X\beta)$$
$$= (y^T - \beta^T . X^T)(y - X\beta)$$
$$= y^T y - (2\beta^T X^T y) + (\beta^T X^T X \beta)$$

Derivating S to minimize error,

$$\frac{dS}{d\beta} = -2X^T y + 2X^T X\beta = 0$$
$$= X^T X \beta = X^T y$$
$$\boxed{\beta = (X^T X)^{-1} X^T y}$$

β's are weights.
x's are features.
y's are labels.

Q2.2 & Q2.3

Performance Metrics for Q2.2 (Lambda = 0)

R^2 on first fold 0.7436109075588095

R^2 on second fold 0.7254814590830649

R^2 on third fold 0.7886997467689636

R^2 on fourth fold 0.763707523461941

R^2 on fifth fold 0.7634663004852206

Mse(Mean Squared Error) on first fold 0.005802855059119702

Mse(Mean Squared Error) on second fold 0.005409484642311184

Mse(Mean Squared Error) on third fold 0.004640479828036911

Mse(Mean Squared Error) on fourth fold 0.003903535811464972

Mse(Mean Squared Error) on fifth fold 0.004172521462143839

Mae(Mean Absolute Error) on first fold 0.056224291026115446

Mae(Mean Absolute Error) on second fold 0.05354179840175395

Mae(Mean Absolute Error) on third fold 0.04918515205473615

Mae(Mean Absolute Error) on fourth fold 0.046740526363488716

Mae(Mean Absolute Error) on fifth fold 0.04870726409938911

Mape(Mean Absolute Percentage Error) on first fold 0.0947800315410192

Mape(Mean Absolute Percentage Error) on second fold 0.09109582121223507

Mape(Mean Absolute Percentage Error) on third fold 0.07999656941493775

Mape(Mean Absolute Percentage Error) on fourth fold 0.07056221917240943

Mape(Mean Absolute Percentage Error) on fifth fold 0.07305565318719245

Performance Metrics for Q2.2(Lambda = 0.01,Learning Rate = 0.0001)

R^2 on first fold 0.017559314536320203

R^2 on second fold 0.016262830086299473

R^2 on third fold 0.021829350444964812

R^2 on fourth fold 0.016867352072022168

R^2 on fifth fold 0.02225579826031987

Mse(Mean Squared Error) on first fold 0.461474482203938

Mse(Mean Squared Error) on second fold 0.4370448114048523

Mse(Mean Squared Error) on third fold 0.3846448025680961

Mse(Mean Squared Error) on fourth fold 0.37308720336769985

Mse(Mean Squared Error) on fifth fold 0.3516782194847648

Mae(Mean Absolute Error) on first fold 0.6686848937277591

Mae(Mean Absolute Error) on second fold 0.6516258030928416

Mae(Mean Absolute Error) on third fold 0.6093203511515451

Mae(Mean Absolute Error) on fourth fold 0.6024878204862381

Mae(Mean Absolute Error) on fifth fold 0.5848848326818956

Mape(Mean Absolute Percentage Error) on first fold 0.9645823708106959

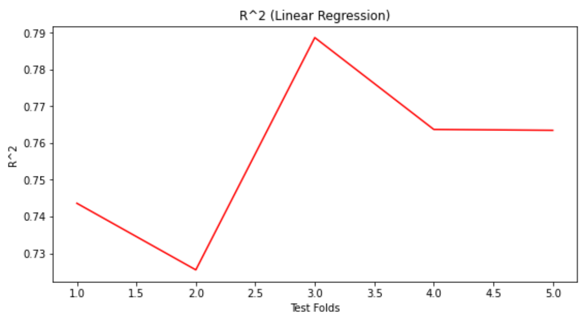
Mape(Mean Absolute Percentage Error) on second fold 0.929813091369729

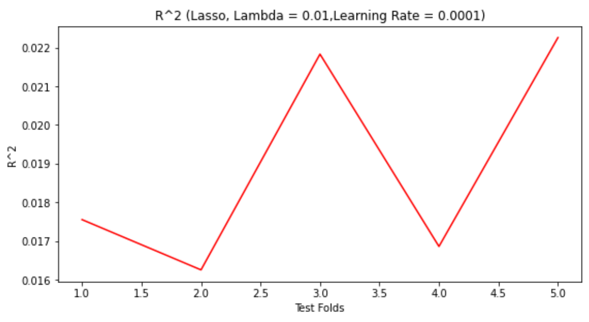
Mape(Mean Absolute Percentage Error) on third fold 0.8416715919277205

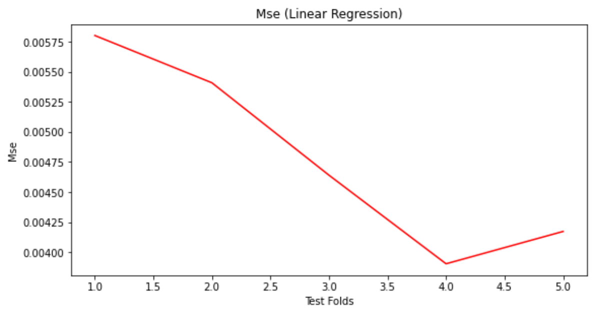
Mape(Mean Absolute Percentage Error) on fourth fold 0.8269462867788901

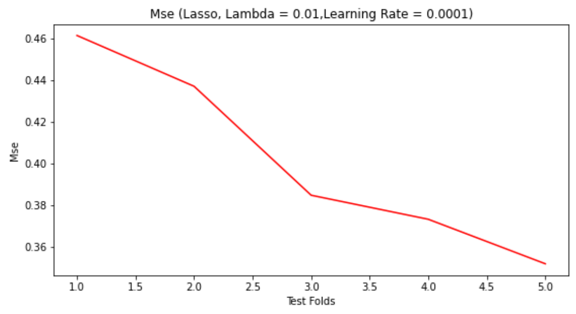
Mape(Mean Absolute Percentage Error) on fifth fold 0.7954080815857988

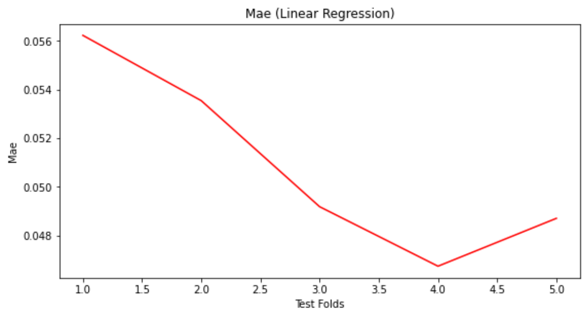
Q2.4

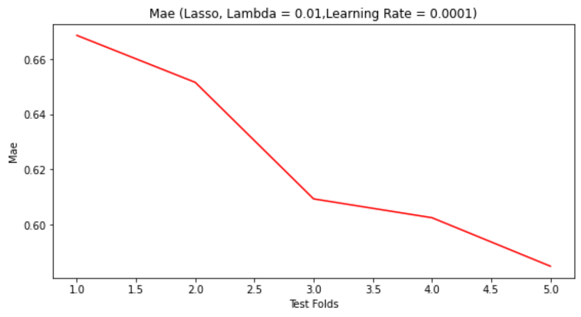


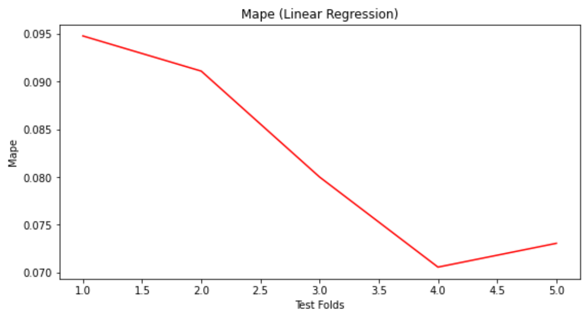


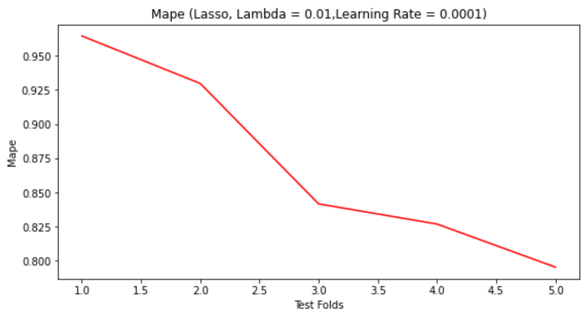












I trained my models with linear regression for the right side. Then, I add lambda = 0.01 as Lasso coefficient and learning rate = 0.0001 to train lasso model with gradient descent. I choose small learning rate to not miss local minima from gradient. Thus, some numerical errors are occurred when we compare the linear regression model.

If we compare Mean Absolute Error (MAE) and Mean Square Error (MSE), we can say that both metric gives similar errors. Lasso model on the right gives better results for the MSE. MAE measures the absolute average distance between the real data and the predicted data, but it fails to punish large errors in prediction. And MSE measures the squared average distance between the real data and the predicted data. So, MSE performs better for reporting the larger error. But MSE also squares up the units of data as well. Therefore, we cannot say MAE or MSE is better than. For this data set, we can clearly see that MSE is similar for both models. Errors are almost same for first 3 fold and it changes after the 3rd fold. Lasso model is penalizing the error. Therefore, after 3rd our metric values are decreasing because of the Lasso.

However, MSE is similar for both models because Lasso model decreases coefficients by a similar amount if we compared with the regression model.

I would select MSE as performance metric because MSE gives higher importance to the error and more sensitive to outliers. MSE takes the squares of the differences between predict and actual which means it give more response than other metrics.

Cross validation based experimental setup helps us to trace each model's performance. If we give fixed dataset to train, we may face overfitting but if we use cross validation technique, we give a chance to our model to train on more new data. So, our model will see different data for each model and creates new predictions based on new data. Cross validation gives a chance to use all of our data. Since we do not have too much data in our dataset, splitting it into fixed training and test set may occur small size of test set. Thus, we may not make assumptions on metrics when we have small test set.

Lasso model's effect on model weights is, lasso forcing weights to decrease or to be zero. Thus, some features and corresponding weights are gone from the model which can occur more understandable models. Similarly, if lasso make some of the features and weights to zero, our model may lose some variable. To predict the chance of admission, our model needs other variables that correlated to the chance of admission. Therefore, L1(lasso) regularization might not be beneficial for the model's performance.

As I mentioned before, if we have a large dataset, we do not need to split our dataset. Because our model will see enough data to predict on test set accurately.

Q3.1

Times are in ms.

```
Mini-Batch Gradient Ascent Algorithm (Size=32)

Learning Rate = 0.0001

Weights at i =  100

[[-0.00969583  0.00817859 -0.02447579  0.01415557  0.01069621  0.0210128

  0.00401557]]

Weights at i =  200

[[-0.0111997   0.00696687 -0.034055    0.01330131  0.01054485  0.02690154

  0.00391274]]

Weights at i =  300

[[-0.01231125  0.00585825 -0.03796945  0.01234936  0.01033461  0.03358913

  0.00396553]]

Weights at i =  400

[[-0.01329973  0.00477357 -0.04172171  0.01118315  0.00992489  0.03355127

  0.00401425]]

Weights at i =  500

[[-0.01413278  0.00373761 -0.04258463  0.01021721  0.00979859  0.03691925

  0.00404699]]

Weights at i =  600
```

```
[[-0.01476122  0.00278158 -0.04349282  0.00899747  0.00951025  0.03654522
   0.00418985]]
Weights at i =  700
[[-0.01561807  0.00178402 -0.04545315  0.00815032  0.00926588  0.03828139
   0.00425508]]
Weights at i =  800
[[-0.01611332  0.00084106 -0.04379711  0.00726735  0.00913727  0.03973865
   0.00435682]]
Weights at i =  900
[[-0.01698291 -0.00012563 -0.04301696  0.0060808   0.00896293  0.04018606
   0.00446995]]
Weights at i =  1000
[[-0.0177622  -0.00117857 -0.04407922  0.00515514  0.00873736  0.03899775
   0.00452869]]
Confusion Matrix(True Positives and False Positives) : 24 23
Confusion Matrix(False Negatives and True Negatives) : 47 89
Accuracy: 63.128491620111724
Precision: 0.5106382978723404
Recall: 0.3380281690140845
Negative Predictive Value(NPV): 0.6544117647058824
False Positive Rate(FPR): 0.20535714285714285
False Discovery Rate(FDR): 0.48936170212765956
F1: 0.4067796610169492
F2: 0.36253776435045315
Training time for Mini Batch Gradient: 0.11798405647277832
--------------------------------------------
Learning Rate = 0.001
Weights at i =  100
[[-0.00700973  0.00596456 -0.04093084  0.00135831  0.00421413  0.04228037
   0.00552664]]
Weights at i =  200
[[-0.01479244 -0.00392998 -0.03397182 -0.01090576  0.00120519  0.03776294
   0.00609449]]
Weights at i =  300
[[-0.02252728 -0.01352203 -0.0487695  -0.02116792 -0.00101151  0.03384311
   0.00682823]]
Weights at i =  400
```

```
[[-0.02880708 -0.02366204 -0.03930456 -0.0334108  -0.00225184  0.04619584
  0.0089969 ]]
Weights at i =  500
[[-0.03583128 -0.03290877 -0.0436907  -0.04401607 -0.00421995  0.04449647
  0.01058923]]
Weights at i =  600
[[-0.04143598 -0.04230741 -0.03981844 -0.05519008 -0.00639661  0.03686124
  0.01142426]]
Weights at i =  700
[[-0.04544087 -0.05100614 -0.03984898 -0.06430155 -0.00843301  0.04534622
  0.01337265]]
Weights at i =  800
[[-0.05053262 -0.05964879 -0.04143943 -0.07379994 -0.01046538  0.04360037
  0.01495502]]
Weights at i =  900
[[-0.05429932 -0.06851638 -0.03996419 -0.08007795 -0.01010343  0.04408229
  0.0161368 ]]
Weights at i =  1000
[[-0.0573737  -0.07774037 -0.04231785 -0.08860965 -0.01186074  0.03625248
  0.01746803]]
Confusion Matrix(True Positives and False Positives) : 24 22
Confusion Matrix(False Negatives and True Negatives) : 47 90
Accuracy: 63.687150837988824
Precision: 0.5217391304347826
Recall: 0.3380281690140845
Negative Predictive Value(NPV): 0.656934306569343
False Positive Rate(FPR): 0.19642857142857142
False Discovery Rate(FDR): 0.4782608695652174
F1: 0.41025641025641024
F2: 0.36363636363636365
Training time for Mini Batch Gradient: 0.11699795722961426
--------------------------------------------
Learning Rate = 0.01
Weights at i =  100
[[-0.07837723 -0.0994679  -0.10905518 -0.08732613 -0.02379883  0.06945466
  0.01501624]]
Weights at i =  200
```

```
[[-0.11565725 -0.19069714 -0.12129694 -0.19303016 -0.05585565  0.10258778
   0.02400143]]
Weights at i =  300
[[-0.12772303 -0.27190815 -0.08372865 -0.28791317 -0.08991093  0.04692995
   0.04966525]]
Weights at i =  400
[[-0.14118096 -0.36003061 -0.15094566 -0.37670216 -0.10339383  0.12168125
   0.07761815]]
Weights at i =  500
[[-0.11211192 -0.43281379 -0.01615988 -0.44968771 -0.11541661  0.14125247
   0.10052892]]
Weights at i =  600
[[-0.11350193 -0.51606963 -0.02145182 -0.50993797 -0.11066017  0.11611003
   0.12410203]]
Weights at i =  700
[[-0.122383   -0.59405087 -0.10120224 -0.57186672 -0.12090155  0.09237106
   0.12533763]]
Weights at i =  800
[[-0.10248906 -0.66911541 -0.02618257 -0.62613084 -0.11339515  0.14177351
   0.12588627]]
Weights at i =  900
[[-0.09677503 -0.7432358  -0.06610042 -0.68509106 -0.12888694  0.07567196
   0.13694788]]
Weights at i =  1000
[[-0.07804099 -0.80794575 -0.09902199 -0.73467152 -0.1334027   0.08844897
   0.15485896]]
Confusion Matrix(True Positives and False Positives) : 22 9
Confusion Matrix(False Negatives and True Negatives) : 49 103
Accuracy: 69.83240223463687
Precision: 0.7096774193548387
Recall: 0.30985915492957744
Negative Predictive Value(NPV): 0.6776315789473685
False Positive Rate(FPR): 0.08035714285714286
False Discovery Rate(FDR): 0.2903225806451613
F1: 0.4313725490196078
F2: 0.3492063492063492
```

Training time for Mini Batch Gradient: 0.12399816513061523

As a discussion, accuracy is almost same for the first two learning rate parameters. Weights are decreasing for the first 2 learning parameters. But then, when we give 0.01 as learning parameter, accuracy and weights are increasing. We pick batches from the current set and our model sees different batches each time. It updates weight after each batch. Thus, we have a better performance than full batch. For the iterations, weights are decreasing for the first two learning parameter but then it is increasing for the last learning parameter because of the gradient ascent. Learning rate 0.01 gives the best accuracy for Mini-Batch Gradient Model.

Stohastic Gradient Ascent Algorithm

Learning Rate = 0.0001

```
Weights at i =  100
[[-0.01391941  0.005698   -0.02146653  0.00340107  0.00254015  0.01819348
  -0.00246336]]
Weights at i =  200
[[-0.01546963  0.00446903 -0.03143397  0.0025629   0.00242715  0.02665285
  -0.00250812]]
Weights at i =  300
[[-0.0166012   0.00335995 -0.03648043  0.00163836  0.00229316  0.03144653
  -0.00248465]]
Weights at i =  400
[[-0.01753621  0.00230602 -0.03932532  0.00066293  0.00214332  0.03428021
  -0.00242889]]
Weights at i =  500
[[-0.01836284  0.00128214 -0.0409962  -0.00034169  0.00198375  0.03600637
  -0.00235519]]
Weights at i =  600
[[-0.01912495  0.00027598 -0.04199239 -0.00136301  0.00181852  0.03707848
  -0.00227086]]
Weights at i =  700
[[-0.0198467  -0.00071917 -0.04258529 -0.00239366  0.00165017  0.03775365
  -0.00217997]]
Weights at i =  800
[[-0.02054216 -0.0017072  -0.04293162 -0.00342917  0.00148027  0.0381839
  -0.00208491]]
Weights at i =  900
[[-0.0212198  -0.00269043 -0.04312519 -0.0044668   0.00130983  0.03846145
  -0.00198714]]
Weights at i =  1000
```

```
[[-0.0218848  -0.0036703  -0.04322337 -0.00550482  0.00113947  0.03864319
  -0.00188756]]
Confusion Matrix(True Positives and False Positives) : 24 23
Confusion Matrix(False Negatives and True Negatives) : 47 89
Accuracy: 63.128491620111724
Precision: 0.5106382978723404
Recall: 0.3380281690140845
Negative Predictive Value(NPV): 0.6544117647058824
False Positive Rate(FPR): 0.20535714285714285
False Discovery Rate(FDR): 0.48936170212765956
F1: 0.4067796610169492
F2: 0.36253776435045315
Training time for Stochastic Gradient: 4.412715196609497
--------------------------------------------
Learning Rate = 0.001
Weights at i =  100
[[-0.01591899 -0.01939426 -0.0438183  -0.00265523  0.00211544  0.03823988
   0.00864831]]
Weights at i =  200
[[-0.02243965 -0.02906915 -0.04346383 -0.01303887  0.00034723  0.03871763
   0.00955013]]
Weights at i =  300
[[-0.02854437 -0.03863928 -0.04285196 -0.02318126 -0.0013091   0.03890246
   0.01048799]]
Weights at i =  400
[[-0.03427532 -0.04811452 -0.04227492 -0.03307836 -0.00285489  0.03908917
   0.01145498]]
Weights at i =  500
[[-0.03964813 -0.05749756 -0.04173384 -0.04273543 -0.00429485  0.03928
   0.01244891]]
Weights at i =  600
[[-0.04467791 -0.066791   -0.04122681 -0.05215805 -0.00563364  0.03947417
   0.01346772]]
Weights at i =  700
[[-0.04937945 -0.07599738 -0.040752   -0.06135198 -0.00687588  0.03967093
   0.0145094 ]]
Weights at i =  800
```

```
[[-0.05376719 -0.08511921 -0.04030765 -0.07032313 -0.00802612  0.03986962
   0.01557202]]
Weights at i =  900
[[-0.05785521 -0.09415895 -0.03989208 -0.07907748 -0.00908878  0.04006962
   0.01665373]]
Weights at i =  1000
[[-0.06165719 -0.10311899 -0.03950369 -0.08762108 -0.0100682   0.04027039
   0.01775276]]
Confusion Matrix(True Positives and False Positives) : 25 22
Confusion Matrix(False Negatives and True Negatives) : 46 90
Accuracy: 64.24581005586593
Precision: 0.5319148936170213
Recall: 0.352112676056338
Negative Predictive Value(NPV): 0.6617647058823529
False Positive Rate(FPR): 0.19642857142857142
False Discovery Rate(FDR): 0.46808510638297873
F1: 0.423728813559322
F2: 0.377643504531722
Training time for Stochastic Gradient: 4.208950519561768
--------------------------------------------
Learning Rate = 0.01
Weights at i =  100
[[-0.04982128 -0.07881479 -0.04422216 -0.08504944 -0.02564894  0.03754078
   0.0108023 ]]
Weights at i =  200
[[-0.07782056 -0.16701178 -0.0410851  -0.16090093 -0.03397282  0.03963391
   0.02257378]]
Weights at i =  300
[[-0.08754712 -0.24896843 -0.03959351 -0.22126342 -0.03709363  0.04153608
   0.03475988]]
Weights at i =  400
[[-0.0867021  -0.32609021 -0.03901516 -0.27058415 -0.03751273  0.04319031
   0.04650933]]
Weights at i =  500
[[-0.07997767 -0.39926258 -0.03893936 -0.31191335 -0.03672364  0.04462729
   0.05737752]]
Weights at i =  600
```

```
[[-0.07015898 -0.46904802 -0.03913366 -0.34728695 -0.03557722  0.04589037
   0.06716558]]
Weights at i =  700
[[-0.05887832 -0.53581824 -0.03946549 -0.37807878 -0.03453855  0.04701619
   0.07581708]]
Weights at i =  800
[[-0.04708055 -0.59983303 -0.03985939 -0.40524031 -0.03384627  0.04803237
   0.08335517]]
Weights at i =  900
[[-0.03530234 -0.66128578 -0.04027298 -0.42945149 -0.03360782  0.04895918
   0.08984507]]
Weights at i =  1000
[[-0.02383868 -0.72032975 -0.04068321 -0.45121406 -0.03385584  0.04981165
   0.09537183]]
Confusion Matrix(True Positives and False Positives) : 26 9
Confusion Matrix(False Negatives and True Negatives) : 45 103
Accuracy: 72.06703910614524
Precision: 0.7428571428571429
Recall: 0.36619718309859156
Negative Predictive Value(NPV): 0.6959459459459459
False Positive Rate(FPR): 0.08035714285714286
False Discovery Rate(FDR): 0.2571428571428571
F1: 0.49056603773584906
F2: 0.40752351097178685
Training time for Stochastic Gradient: 4.48769474029541
```

For the stochastics gradient, accuracy is increasing as learning parameter increased. The learning parameter 0.01 gives the best accuracy for the stochastic gradient. For the learning rates ,0.0001 and 0.001 weights are decreasing per 100 iteration but for the learning rate 0.01 weights decreases for the beginning iterations, then increases. As a difference from mini batch gradient, stochastic gradient updates weights for each label per iteration and gives better accuracy. We can say that weights are changing according to the learning rates and results from the gradient. Bigger learning rate means bigger impact of the gradient. The main difference than batch gradient is, stochastic gradient using a single sample each time rather than giving batches.

Q3.2

Full Batch Gradient Ascent Algorithm

```
Learning Rate = 0.0001

Weights at i =  100

[[-0.0073864   0.00268685 -0.03226596 -0.01212498  0.00950136  0.01752604

  0.00652588]]

Weights at i =  200

[[-0.01148317 -0.00156744 -0.03271945 -0.01448954  0.00895156  0.01809152

  0.00732911]]

Weights at i =  300

[[-0.01542638 -0.00578222 -0.03236535 -0.01681317  0.00842192  0.01809603

  0.00814486]]

Weights at i =  400

[[-0.01928033 -0.00997595 -0.03199109 -0.01909185  0.00791435  0.01808134

  0.00896554]]

Weights at i =  500

[[-0.02304893 -0.01414953 -0.03162391 -0.02132629  0.00742841  0.01806642

  0.00979071]]

Weights at i =  600

[[-0.02673398 -0.01830325 -0.03126459 -0.02351735  0.00696362  0.01805192

  0.01062019]]

Weights at i =  700

[[-0.03033716 -0.02243738 -0.03091296 -0.0256659   0.00651947  0.01803785

  0.01145383]]

Weights at i =  800

[[-0.03386014 -0.02655219 -0.03056886 -0.02777282  0.00609548  0.01802416

  0.01229144]]

Weights at i =  900

[[-0.03730457 -0.03064793 -0.0302321  -0.02983893  0.00569116  0.01801085

  0.01313287]]

Weights at i =  1000

[[-0.04067205 -0.03472486 -0.0299025  -0.0318651   0.00530604  0.01799788

  0.01397796]]

Confusion Matrix(True Positives and False Positives) : 19 16

Confusion Matrix(False Negatives and True Negatives) : 52 96
```

```
Accuracy: 64.24581005586593

Precision: 0.5428571428571428

Recall: 0.2676056338028169

Negative Predictive Value(NPV): 0.6486486486486487

False Positive Rate(FPR): 0.14285714285714285

False Discovery Rate(FDR): 0.45714285714285713

F1: 0.3584905660377358

F2: 0.2978056426332288

Training time for Full-Batch Gradient: 0.1119990348815918

--------------------------------------------

Learning Rate = 0.001

Weights at i =  100

[[-0.03169021 -0.05650344 -0.03013514 -0.01143774 -0.00360142  0.01785041

   0.01843858]]

Weights at i =  200

[[-0.06303702 -0.09611784 -0.02706821 -0.03156663 -0.00685686  0.01774702

   0.02670803]]

Weights at i =  300

[[-0.08793474 -0.13406669 -0.02458801 -0.04827496 -0.00854086  0.01766429

   0.03523054]]

Weights at i =  400

[[-0.10765247 -0.17055891 -0.02256985 -0.06223317 -0.00902615  0.01759258

   0.04388306]]

Weights at i =  500

[[-0.12322382 -0.20576755 -0.02091695 -0.07398767 -0.0086134   0.01752732

   0.05256752]]

Weights at i =  600

[[-0.13548021 -0.23983411 -0.01955422 -0.08397475 -0.00754106  0.01746638

   0.06120694]]

Weights at i =  700

[[-0.14508656 -0.27287385 -0.01842315 -0.09253965 -0.00599652  0.01740877

   0.06974175]]

Weights at i =  800

[[-0.15257357 -0.30498079 -0.01747795 -0.09995484 -0.00412631  0.01735407

   0.07812632]]

Weights at i =  900

[[-0.15836486 -0.33623203 -0.01668257 -0.10643549 -0.00204461  0.01730208
```

```
  0.08632625]]
Weights at i =  1000
[[-1.62798912e-01 -3.66691194e-01 -1.60085031e-02 -1.12152019e-01
   1.59913991e-04  1.72527264e-02  9.43161003e-02]]
Confusion Matrix(True Positives and False Positives) : 19 7
Confusion Matrix(False Negatives and True Negatives) : 52 105
Accuracy: 69.27374301675978
Precision: 0.7307692307692307
Recall: 0.2676056338028169
Negative Predictive Value(NPV): 0.6687898089171974
False Positive Rate(FPR): 0.0625
False Discovery Rate(FDR): 0.2692307692307692
F1: 0.3917525773195876
F2: 0.3064516129032258
Training time for Full-Batch Gradient: 0.11996722221374512
--------------------------------------------
Learning Rate = 0.01
Weights at i =  100
[[-0.24824348 -0.39254212 -0.20051328 -0.32741436 -0.13764348  0.37415645
   0.11437021]]
Weights at i =  200
[[-0.36721728 -0.75465466 -0.24040019 -0.61895721 -0.24191201  0.76603358
   0.224859  ]]
Weights at i =  300
[[-0.41694961 -1.09418273 -0.21832483 -0.87772193 -0.3364923   0.33913189
   0.33190745]]
Weights at i =  400
[[-0.46590347 -1.42705984 -0.50518929 -1.09920208 -0.41882602  0.30145975
   0.42858325]]
Weights at i =  500
[[-0.45027988 -1.73960428 -0.44188185 -1.28051109 -0.4744253   0.17566365
   0.52568518]]
Weights at i =  600
[[-0.44396478 -2.0463738  -0.63005381 -1.44987186 -0.53193126 -0.10500407
   0.61340121]]
Weights at i =  700
[[-0.39547687 -2.33479627 -0.29552901 -1.60878116 -0.58679667  0.09114106
```

```
  0.69973704]]
Weights at i =  800
[[-0.37220696 -2.62323812 -0.40315837 -1.75932203 -0.6388247   0.21508126
  0.77834436]]
Weights at i =  900
[[-0.33643012 -2.89974004 -0.36955734 -1.88339266 -0.67684927  0.09175303
  0.84821617]]
Weights at i =  1000
[[-0.29273981 -3.16688804 -0.32966615 -2.0049098  -0.71581624  0.29541433
  0.91537398]]
Confusion Matrix(True Positives and False Positives) : 22 9
Confusion Matrix(False Negatives and True Negatives) : 49 103
Accuracy: 69.83240223463687
Precision: 0.7096774193548387
Recall: 0.30985915492957744
Negative Predictive Value(NPV): 0.6776315789473685
False Positive Rate(FPR): 0.08035714285714286
False Discovery Rate(FDR): 0.2903225806451613
F1: 0.4313725490196078
F2: 0.3492063492063492
Training time for Full-Batch Gradient: 0.11499953269958496
```

As a discussion, similarly full- batch gradient gives best accuracy when learning rate equals to the 0.01. Because of the gradient's impact occurred by learning rate. It is separated from mini-batch gradient by taking a full batch rather than size = 32 batch. In other words, we are giving the whole data rather than dividing it into samples. Weights are decreasing for the first two learning rates per iteration, but for the last learning rate, weights start to decrease but then, they start to increase which concludes an increase in accuracy.

Q3.3

I checked the weights after each iteration and training for each model and we can see that some of the features are less important than the others. We can compare the models by comparing the absolute values of weights. Feature normalization was needed for feature importance because some features like gender and port of embarkation were not in the same type of others (categorical and continuous). Therefore, I changed them the integer numbers to make them possible to calculate weights and test them on test data. After the normalization, I tried to compare the weights from the full batch gradient and I realized that "Gender" and "Siblings" features are the most important features according to their absolute value. Finally, I realized that batch size = 32 model is slower than the full batch model. Full batch model gives better performance than mini-batch model.