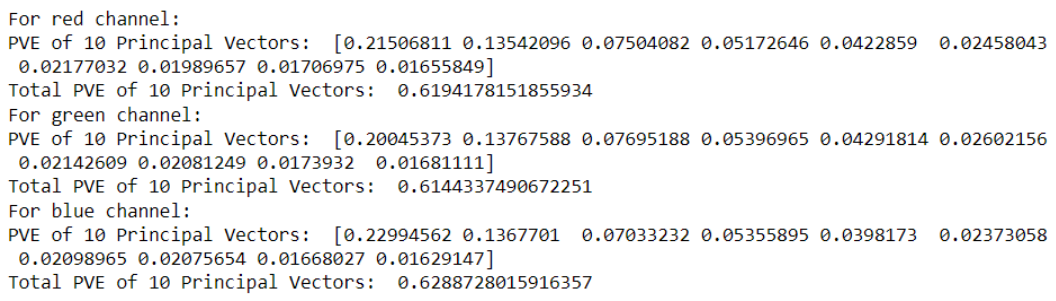
Berk Saltuk Yılmaz
21903419
CS464 - 02 - Homework 2

Question 1)

Q1.1)

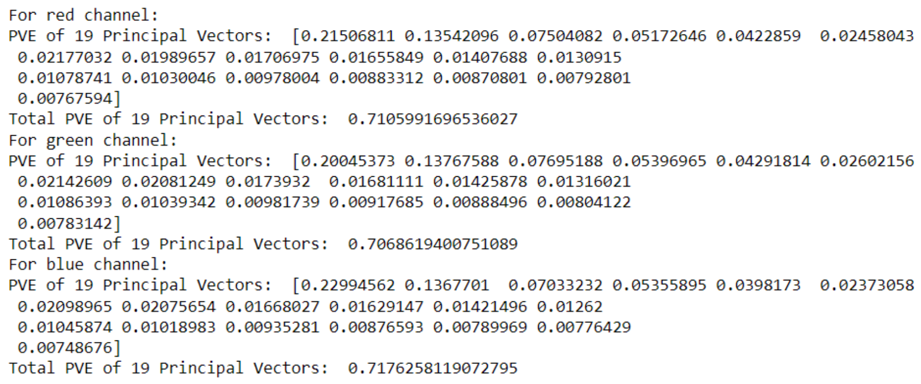
As can be seen from the following figure, the first ten principal components explain 61.9%, 61.4%, and 62.8% of the variance for the red, green, and blue channels, respectively.



```
For red channel:
PVE of 10 Principal Vectors:  [0.21506811 0.13542096 0.07504082 0.05172646 0.0422859  0.02458043
 0.02177032 0.01989657 0.01706975 0.01655849]
Total PVE of 10 Principal Vectors:  0.6194178151855934
For green channel:
PVE of 10 Principal Vectors:  [0.20045373 0.13767588 0.07695188 0.05396965 0.04291814 0.02602156
 0.02142609 0.02081249 0.0173932  0.01681111]
Total PVE of 10 Principal Vectors:  0.6144337490672251
For blue channel:
PVE of 10 Principal Vectors:  [0.22994562 0.1367701  0.07033232 0.05355895 0.0398173  0.02373058
 0.02098965 0.02075654 0.01668027 0.01629147]
Total PVE of 10 Principal Vectors:  0.6288728015916357
```

(Figure 1: PVE for first ten PCs)

After experimenting with different k values, I concluded that the first nineteen principal components are required to obtain at least 70% PVE for each channel. Even though eighteen principal components were enough for red and blue channels, the green channel had a PVE that is less than 70% with eighteen PCs. The first nineteen PCs and corresponding PVE values can be seen in the next figure.

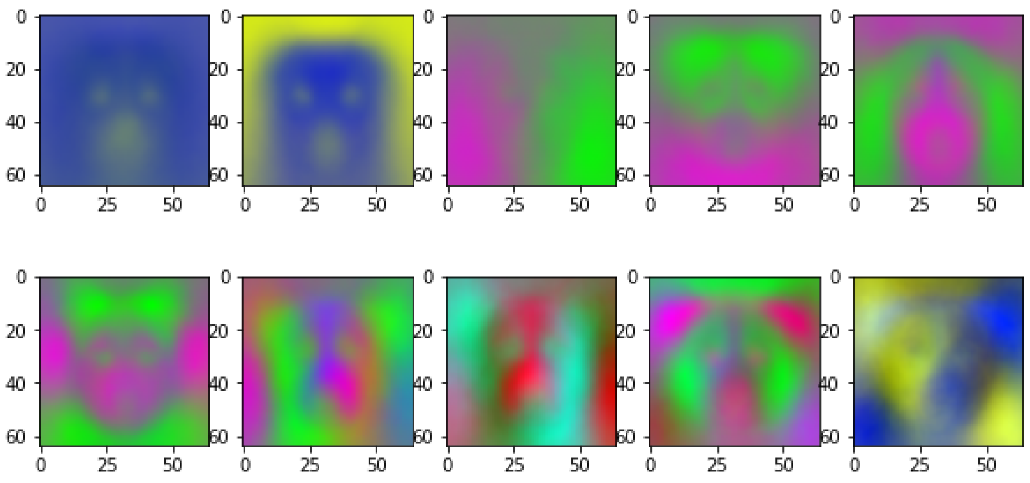


```
For red channel:
PVE of 19 Principal Vectors:  [0.21506811 0.13542096 0.07504082 0.05172646 0.0422859  0.02458043
 0.02177032 0.01989657 0.01706975 0.01655849 0.01407688 0.0130915
 0.01078741 0.01030046 0.00978004 0.00883312 0.00870801 0.00792801
 0.00767594]
Total PVE of 19 Principal Vectors:  0.7105991696536027
For green channel:
PVE of 19 Principal Vectors:  [0.20045373 0.13767588 0.07695188 0.05396965 0.04291814 0.02602156
 0.02142609 0.02081249 0.0173932  0.01681111 0.01425878 0.01316021
 0.01086393 0.01039342 0.00981739 0.00917685 0.00888496 0.00804122
 0.00783142]
Total PVE of 19 Principal Vectors:  0.7068619400751089
For blue channel:
PVE of 19 Principal Vectors:  [0.22994562 0.1367701  0.07033232 0.05355895 0.0398173  0.02373058
 0.02098965 0.02075654 0.01668027 0.01629147 0.01421496 0.01262
 0.01045874 0.01018983 0.00935281 0.00876593 0.00789969 0.00776429
 0.00748676]
Total PVE of 19 Principal Vectors:  0.7176258119072795
```

(Figure 2: PVE for first nineteen PCs)

Q1.2)

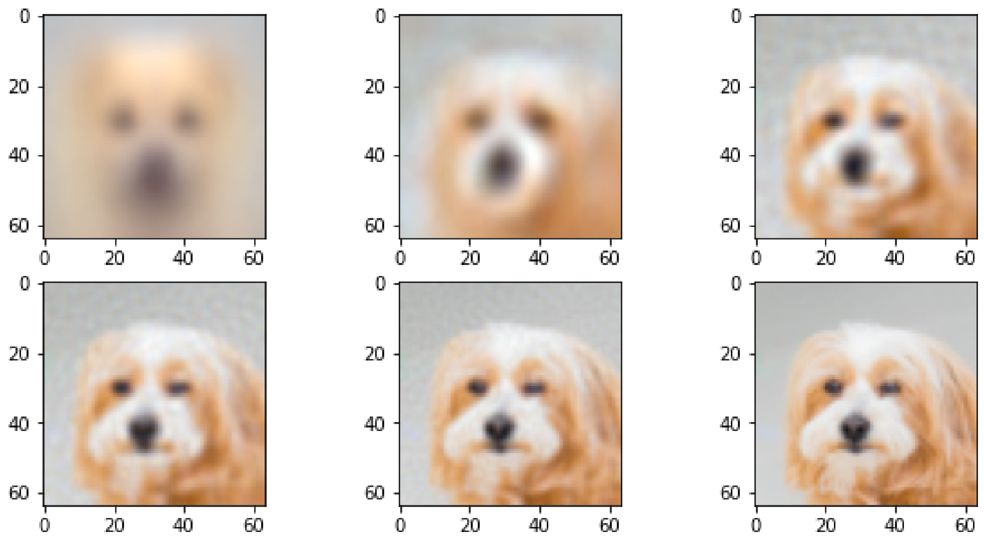
After completing the steps, I obtained visuals for the first ten eigenvectors of dog images:


(Figure 3: Visuals of eigenvalues)

As I was expecting, these visuals are like the infrastructure of a dog image; they carry certain characteristics of dog images 🙂.

Q1.3)

It is possible to reconstruct an original dog image using principal components: after obtaining the principal components, we can find the projection (dimension-reduced X values) of X by having the dot product of mean-centered X and principal components array. Then, we can have a dot product of the projection and principal components array and add the mean we have subtracted in the first place. Here you go, the reconstructed image for k values {1, 50, 250, 500, 1000, 4096}!

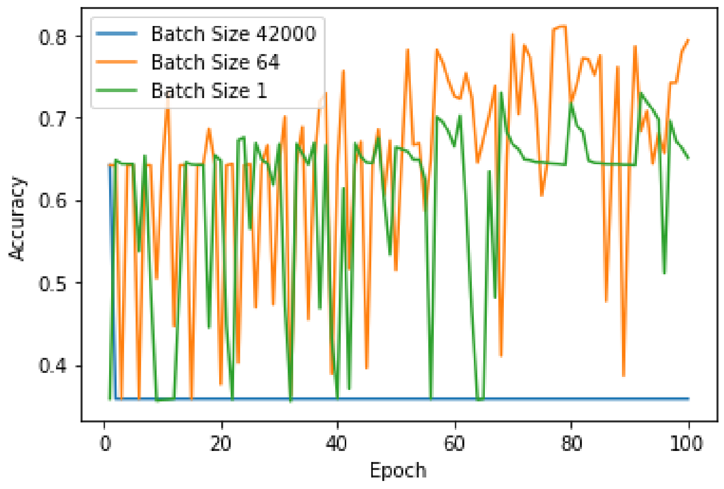

(Figure 4: Reconstruction with different k values)

As you may notice, as the principal components used in the reconstruction increase, we get an image with higher quality (and it is cool to see how similar the first image and the visual of the first principal component are).

Question 2)

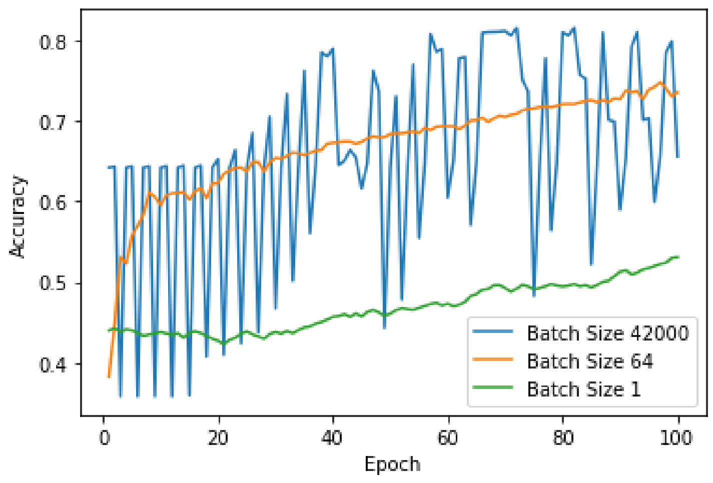
Q2.1)

I trained three logistic regression models with batch sizes 42000, 64, 1. First, I trained the models with a learning rate of 1 but since the full-batch method was resulting in an overflow error, I also trained these models with 0.01 learning rate. As can be seen in the following image, full-batch was resulted in very low accuracy whereas mini-batch and stochastic models performed better. However, mini-batch had higher accuracies overall when the learning rate is 1.



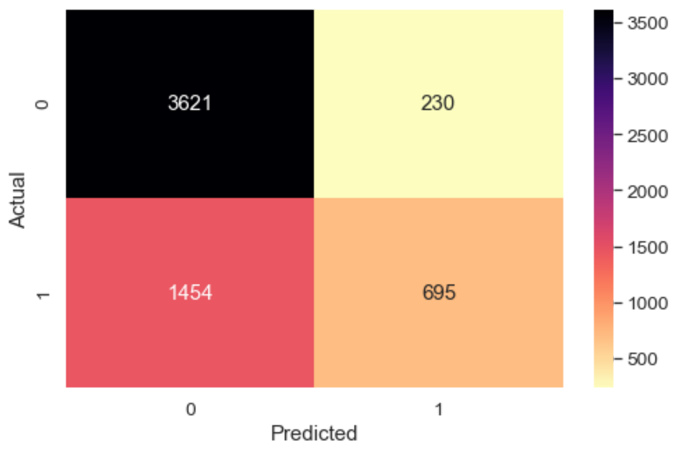
(Figure 5: Accuracy vs. Epoch w/learning rate = 1)

On the other hand, when the learning rate is 0.01, there are no overflow errors and both mini-batch and full-batch are performing better than stochastic gradient ascent. However, mini-batch is more stable overall and the final accuracy after 100 epochs is higher than full-batch. Therefore, mini-batch seems to be the optimal. Moreover, when the batch size decreases the training time decreases dramatically. The mini-batch and stochastic models are trained in seconds where full-batch is trained in minutes.



(Figure 6: Accuracy vs. Epoch w/learning rate = 0.01)

Confusion matrix for the logistic regression model with mini-batch is the following:

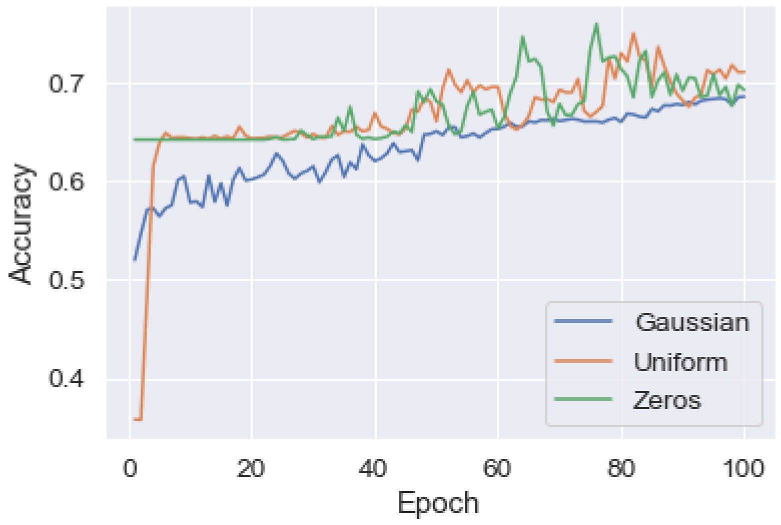


(Figure 7: Confusion Matrix for LR with mini-batch)

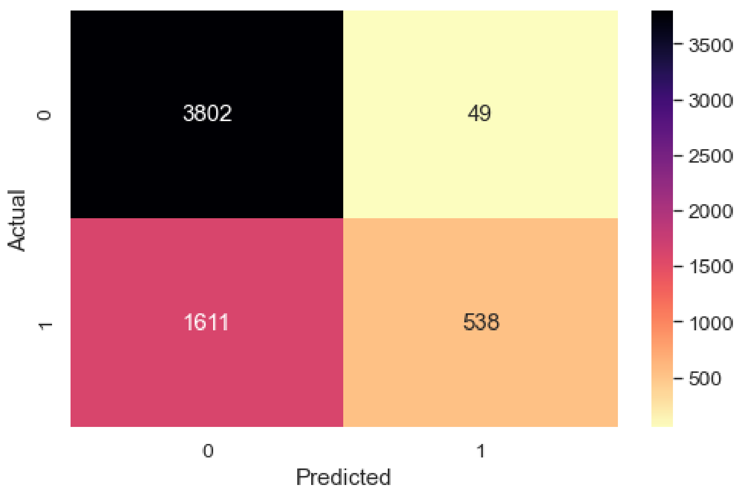
Q2.2)

Different initialization techniques resulted in similar but different accuracy rates. Initialization with zeros and from uniform distributions were nearly always performed better than initialization from Gaussian distribution. However, the effect of initialization type is not as significant as the batch size.

After 100 epochs, initialization from uniform distribution had a higher accuracy.



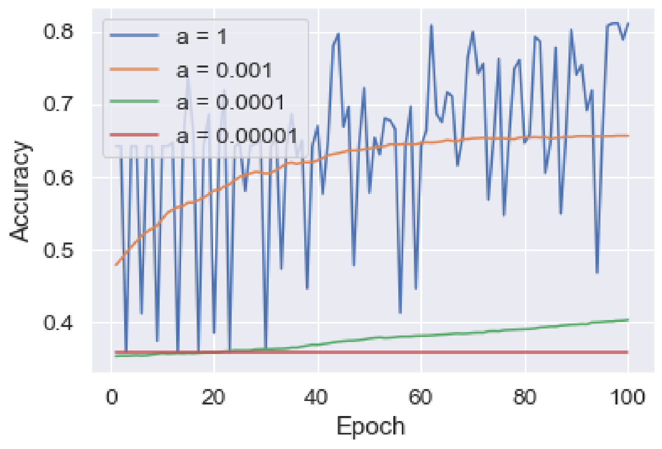
(Figure 6: Accuracy vs. Epoch for Different Init. Types w/learning rate = 0.01)



(Figure 7: Confusion Matrix for mini-batch with uniform init.)

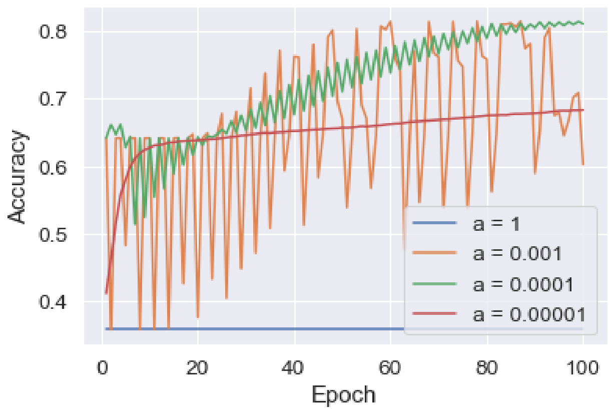
Q2.3)

I trained models using batch size 64 (which I found optimal in 2.1) for different learning rates α ∈ {1, 10**−3 , 10**−4 , 10**−5} and initialized weights from Gaussian distribution. The best learning rate was 1. However, I would prefer 10**-3 since it is more continuous and stable. Moreover, I believe that for a larger number of epochs, this learning rate will perform even better. If we choose the learning rate extensively too large, the model may converge too quickly but the solution may not be correct. and high learning rates results in sudden changes as in this example. On the other hand, too small learning rates may cause the model to adapt too slowly.



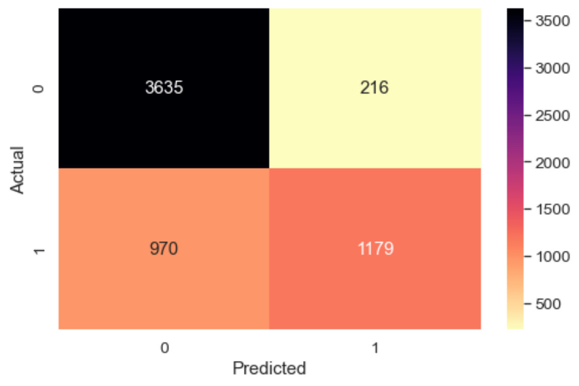
(Figure 8: Accuracy vs Epoch, mini batch for different learning rates)

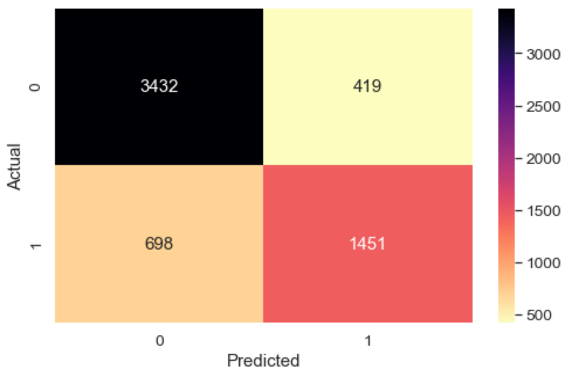
Moreover, even if it was asked to choose the optimal batch size, I also tried full-batch gradient ascent with different learning rates. Even though it performed worse than mini_batch gradient ascent for learning rates 1 and 0.01, it had a higher accuracy for the learning rate 10**-4.



(Figure 8: Accuracy vs Epoch, full batch for different learning rates)

I am providing confusion matrices for both of "optimal" models.





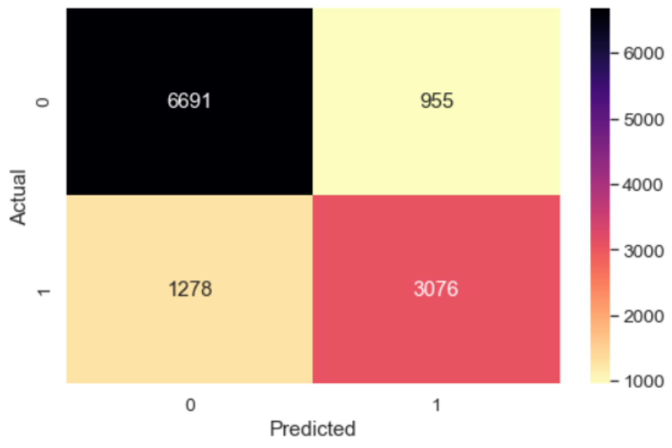
(Figure 9: Conf. Matrix for Mini-batch) (Figure 10: Conf. Matrix for full-batch)

Q2.4)

It is not easy to choose an optimal model in such a sensitive case since labeling unstable grids as stable may cause problems by harming people, therefore; it can be said that false positive number should be lower in this case. I have considered three different "optimal-ish" models for this problem:

1) logistic regression model with mini-batch (batch size = 64), weights from uniform distribution, and learning rate 1

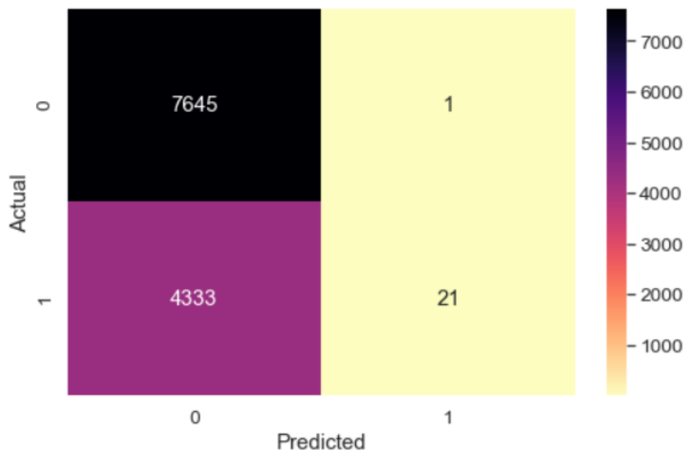
```
Accuracy:  0.8139166666666666
Label 0: Accuracy 0.8750980905048391
Label 1 Accuracy:  0.7064768029398254
Precision:  0.7630860828578516
Recall:  0.7064768029398254
F1:  0.7336911150864639
F2:  0.7171166130461137
F0.5:  0.7510499072175018
FPR:  0.12490190949516086
```



(Figure 11: Optimal 1)

2) logistic regression model with mini-batch (batch size = 64), weights from uniform distribution, and learning rate 10**-3

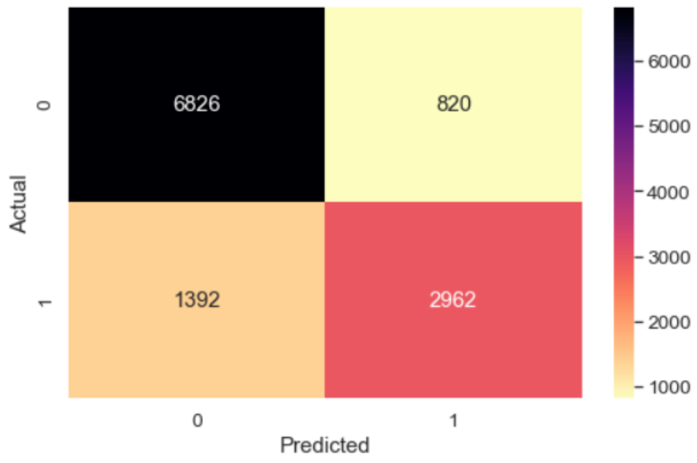
```
Accuracy:  0.6388333333333334
Label 0: Accuracy 0.9998692126602144
Label 1 Accuracy:  0.00482315112540193
Precision:  0.9545454545454546
Recall:  0.00482315112540193
F1:  0.009597806215722121
F2:  0.006021332721642391
F0.5:  0.02363800090049527
FPR:  0.00013078733978550875
```



(Figure 12: Optimal 2)

3) logistic regression model with full-batch, weights from uniform distribution, and learning rate 10**-4

```
Accuracy:  0.8156666666666667
Label 0: Accuracy 0.8927543813758828
Label 1 Accuracy:  0.6802939825447865
Precision:  0.7831835007932311
Recall:  0.6802939825447865
F1:  0.7281219272369714
F2:  0.6986508161147278
F0.5:  0.7601888923108511
FPR:  0.10724561862411719
```



(Figure 13: Optimal 3)

Considering time efficiency, accuracy, FPR and f-beta scores, I believe the optimal model is the model 1. Model 2 has the lowest FPR which is important in our case since false positives may have bad consequences. However, the f-beta scores are too low since this model is pretty stingy while labeling as stable. The third model is pretty accurate has a relatively low FPR (it is not that reliable since it marks 1 out of 10 grids stable even if they are not), however; it is too slow and not very scalable if we increase epoch number. The first model is pretty accurate has relatively low FPR and has higher f-beta scores compared to other models. This makes it the optimal in this case even if it is not the ideal model.