Tobias Fühles, 11936325

Lab assignment – Foundations of Data Analysis

Task 1 – Visualize the data

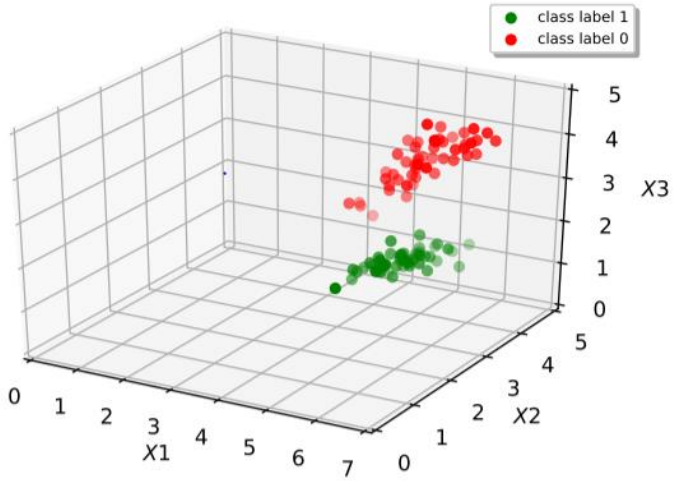

Figure 1 Visualization of "lab_iris_data.csv"

Task 2 – Implement and train a regularized logistic regression model using stochastic gradient descent

Functions and parameters have been defined accordingly to the assignment sheet and can be found in the corresponding .py file.

To facilitate the evaluation the code is exemplary executed and described in the following:
The functions "logistic", "create_ind", "grad" and "train_classifier" are defined and implemented. Because for each iteration of "train_classifier" k random sample and label values are used, np.random.seed(1) is executed to initialize the RandomState.

Result of train_classifier(100, 0.1, 20):

```
[matrix([[ 0.02875],
         [ 0.04525],
         [-0.04075],
         [ 0.01   ]]),
 matrix([[-0.04445836],
         [ 0.02744976],
         [-0.11900035],
         [-0.00020339]]),
 matrix([[-0.00903448],
         [ 0.06343447],
         [-0.12821226],
         [ 0.0094752 ]]),
```

Figure 2 Results 1-3

```
[matrix([[ 0.05398133],
         [ 0.32005777],
         [-0.52279202],
         [ 0.05280776]]),
 matrix([[ 0.04939837],
         [ 0.32009505],
         [-0.52926375],
         [ 0.05235822]]),
 matrix([[ 0.05071729],
         [ 0.32281693],
         [-0.53317649],
         [ 0.05299461]])]
```

Figure 3 Results 49-51

```
matrix([[ 0.06587632],
         [ 0.41129864],
         [-0.68091442],
         [ 0.06669778]]),
matrix([[ 0.06758966],
         [ 0.41373633],
         [-0.68259739],
         [ 0.0671947 ]]),
matrix([[ 0.06874355],
         [ 0.41577833],
         [-0.68444962],
         [ 0.06759922]])]
```

Figure 4 Results 98-100

To check if this results in a well-trained model, the loss (for respective functions see .py file) was determined and ploted (figure 5) over the iterations of t.

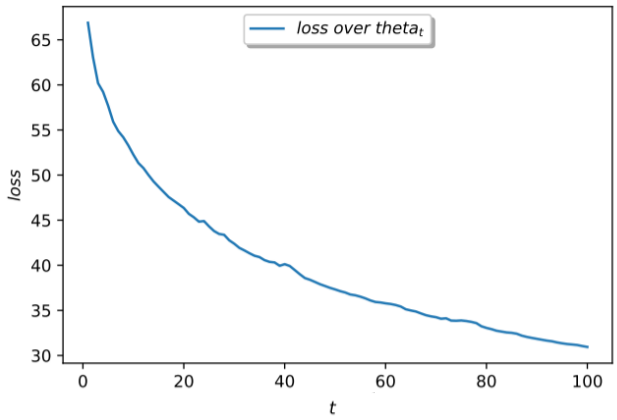

Figure 5 Loss over generated theta values

Figure 5 shows the loss with a specific amount of "wiggle", as a result of our batch size (k = 20). The curve shows the format of a good learning rate, therefore we can conclude that the choice of the parameters (batch size = 20, initial learning rate = 0.1, number of iterations = 100) results in a well-trained model.

To evaluate the classifier in terms of speed and quality the learning rate was generated with various parameters. In the following one parameter (number of iterations, initial learning rate and batch size) is adjusted after another while keeping the remaining parameters fixed. Before each new execution of the train_classifier () function to determine theta again np.random.seed(1) was executed.

Firstly, the number of iterations was adapted to 50 and 1000.

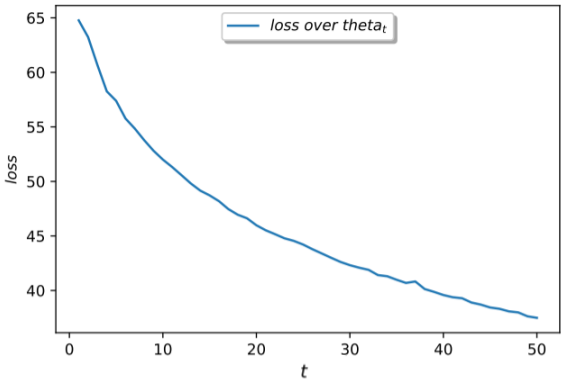

Figure 6 Change of number of iterations from 100 to 50

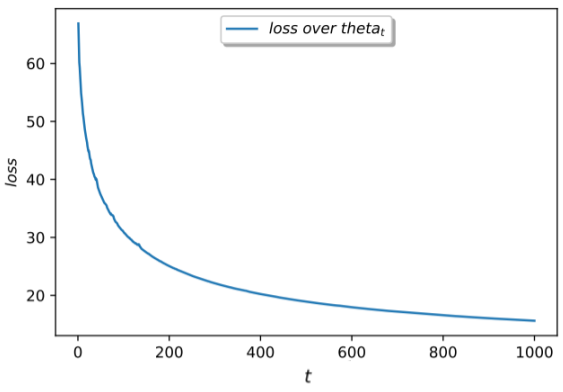

Figure 7 Change of number of iterations from 100 to 1000

As figure 6 shows a reduction of the number of iterations results in a worse learning rate and respectively in a worse trained model. On the other hand, figure 7 shows an improvement of the learning rate by increasing the number of iterations to 1000. By this means the model is improved but at the expense of computational time which was increased from almost real time to 7 sec. for T = 1000.

Secondly, the initial learning rate was adapted to 0.01 and 1

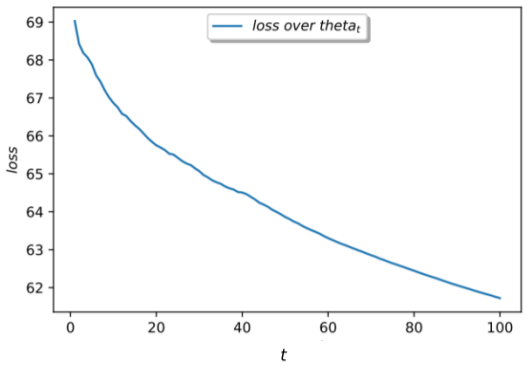

Figure 8 Change of initial learning rate from 0.1 to 0.01

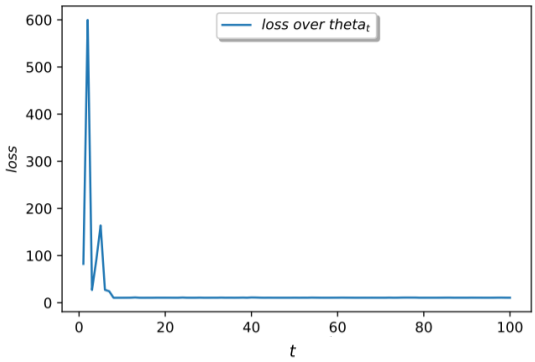

Figure 9 Change of initial learning rate from 0.1 to 1

As figure 8 shows a reduction of the learning rate to 0.01 does not result in an improved learning rate or model with the loss still having a value around 60 for 100 iterations and batch size 20. As a smaller learning rate is usually better in terms of exactly finding the minima this is apparently only achievable with a higher number of iterations.

As a learning rate of 1 (figure 9) already appears intuitively too high, this thought is confirmed by the python implementation. The learning rate first oscillates probably due to diverging weights before being stuck at 0. An increase of the learning rate from 0.1 seems therefore without further use, except experimenting with values from 0.1 to 0.5.

Thirdly, the batch size was adapted from 20 to 5 and 100.

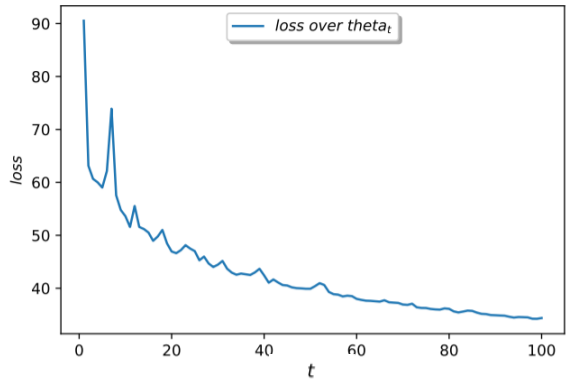

Figure 10 Change of batch size from 20 to 5

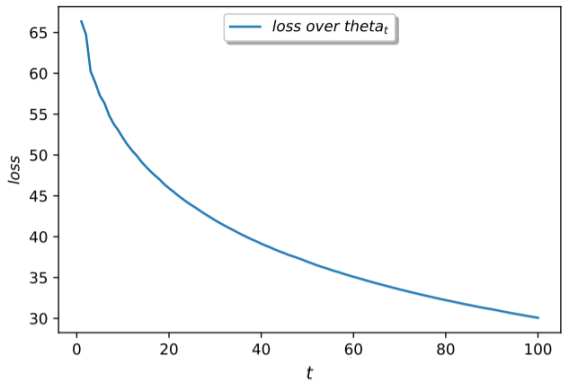

Figure 11 Change of batch size from 20 to 100

Due to the decrease of the batch size to 5 in figure 5 the "wiggle" increases, because other than in figure 11 where the batch size matches with the training data, the gradient is not updated monotonically. Based on the value of the loss reached for 100 iterations, a batch size of 100 is to be preferred over an decrease to 5. Never the less 20 seems a good choice for the batch size as this also yields in a loss around 30 for 100 iterations.

Task 3 - Plot the separating hyperplane

Visualization of the data points and the separating hyperplane from different angles.

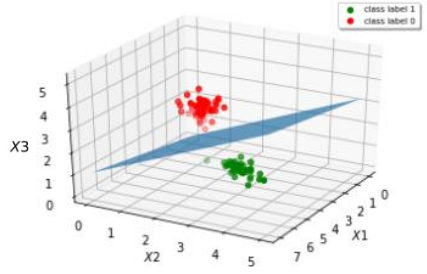

Figure12 Hyperplane visualization 1

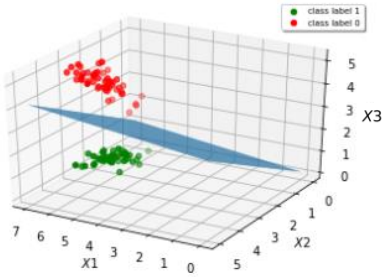

Figure 13 Hyperplane visualization 2

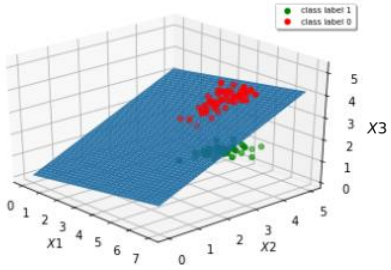

Figure 14 Hyperplane visualization 3

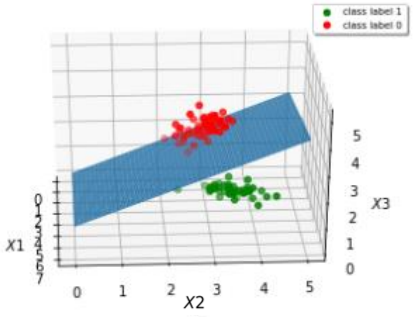

Figure 15 Hyperplane visualization 4

To determine the hyperplane the hypothesis was solved for X3 (see .py file) with the most iterated theta value at t = 100:

```
matrix([[ 0.06874355],
        [ 0.41577833],
        [-0.68444962],
        [ 0.06759922]])]
```

Task 4 – Further questions

Given the above Note in Task 2 about non-stochastic Gradient Descent, why do you think the different variations of the algorithm exist, i.e. when is one more useful than the other?

For non-stochastic Gradient Descent, a run through the complete dataset is necessary to perform a single update on the parameter (e.g. theta), therefore this approach is computational intense for a large dataset and may take too long or needs to much computational resources. Stochastic Gradient Descent is a faster way to update the parameter(s), because it is only using some values of the dataset and starts right away with the improvement. On the other side Stochastic Gradient Descent does not minimize the loss function as good as non-stochastic Gradient Descent but is a reasonably good alternative due to the superiority in computational efficiency.

Under what conditions do they produce the same trained model?

In case the Stochastic Gradient Descent chooses a batch size equivalent to the amount of values in the dataset the result will be the same as for non-stochastic Gradient Descent.

Tobias Fühles, 11936325

How (if at all) would you change the pseudocode in Algorithm 1 to use a different loss function?

For the use of another loss function (e.g. "squared loss") there would be no need to take a random batch size, because the batch size would have to match with the data set in order to gain the best separating hyperplane (a random choice should also result in an optimization though). Also, I would take the derivative not from theta but the weight parameter w(j) and optimate this with the adjusted algorithm.

How does the regularized model we used here differ from the result if we had not used a regularization term?

A regularized model can minimize the risk of overfitting, by reducing the variance. Here we used a regularization factor of 1, but increasingly the regularization factor excessively can result in a rising bias and therefore in the creation of underfitting.