MLVC Assignment1 Report

Jakob Troidl, Lucas da Cunha Melo, Martin Ptacek

November 2019

1 Binary Classification and the Perceptron

To demonstrate binary classification using a perceptron, the MNIST dataset is used in the following experiments. It consists of grayscale images of handwritten digits from zero to nine, as shown in Figure 1a. Only the digits zero and one are used in this binary classification exercise, with 200 of each (totaling 400) used for training, and 200 (totaling 400) used for testing.

The images are then binarized by setting the values of pixels with intensity larger than zero to one, as shown in Figure 1b. This operation allows for the extraction of region properties; in this case: filled area and solidity[1]. These properties are then used as features for the training and testing of a perceptron; Figure 2 shows their plot.

Two versions of the perceptron algorithm were implemented: *online* and *batch*. For the first experiment, the homogeneous coordinates of the input data were used, producing an augmented weight vector after training. The decision

(a) A subset of the MNIST dataset

(b) The grayscale zero and one (left) are binarized (right). Pixels with intensity larger than zero are set to one

Figure 1: Data preparation and feature extraction

[1] https://www.mathworks.com/help/images/ref/regionprops.html

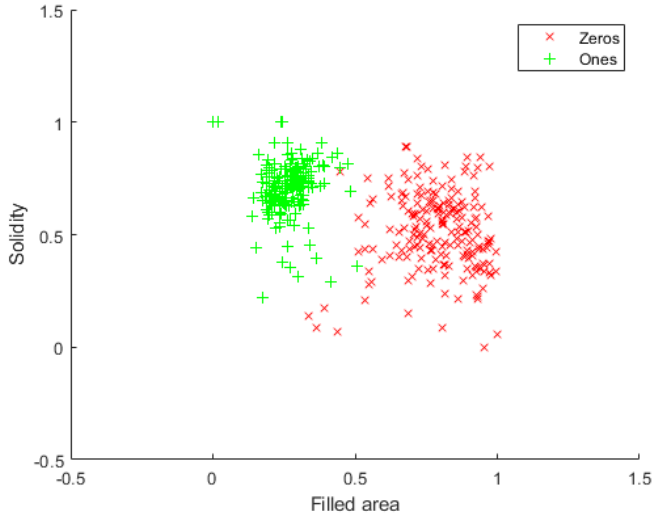

Figure 2: The two features extracted: filled area and solidity

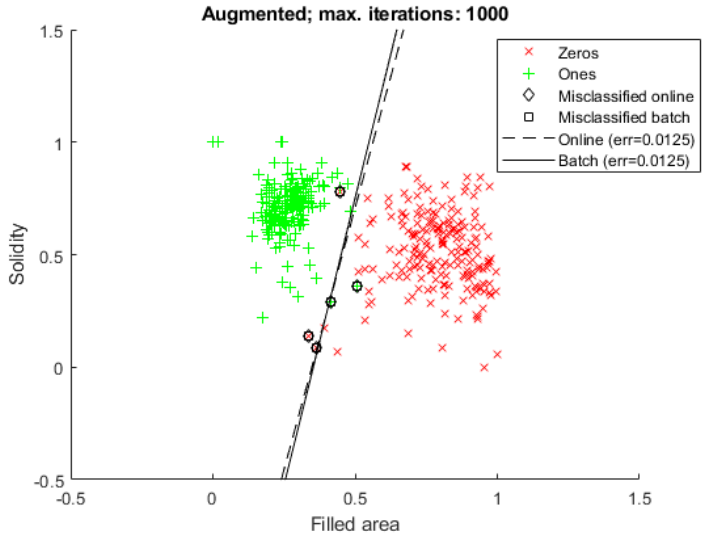

Figure 3: Decision boundaries of the *augmented* models. The misclassified samples are highlighted

boundary for both algorithms is shown in Figure 3, together with the misclassified samples for each perceptron version and their error rate against the training set itself. Visually, both boundaries are very similar, and they also delivered an identical classification of the training set.

Figure 4: Decision boundaries of the *transformed* models. The misclassified samples are highlighted

It became visible in Figure 2 that the data used is not linearly separable. The perceptron training algorithm is not capable of detecting whether the input data is linearly separable or not, as the lack of convergence for non linearly separable data can behave similarly as the slow-converging training of linearly separable data.

For the second experiment, the transformation

$$\Phi(\mathbf{x}) : (x_1, x_2) \rightarrow (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)$$

is used on the initial features, resulting in six-dimensional data points. They were used as training input and the output weights were used to plot a decision boundary, shown in Figure 4. The *batch* version was able to separate the training data more accurately than the *online* version.

In the third and last experiment, instead of using features extracted from the input images, the image pixels were used as features. The images ($n \times n$ matrices) were flattened into vectors (of size n^2), which were then input into both training algorithms, resulting in a weight vector of size n^2. To visualize the weight vector, it was transformed back into a $n \times n$ image, shown in Figure 5.

To conclude the experiments, the *online* and *batch* models of the three variants (*augmented*, using homogeneous coordinates; *transformed* with $\Phi(\mathbf{x})$; and *pixelwise*) were tested using the test image set. The error rates of each of the six models were calculated and are shown in Figure 6. The *augmented* models delivered identical error rates, which is expected, considering the similarity of the decision boundary shown in Figure 3. The *transformed* models delivered

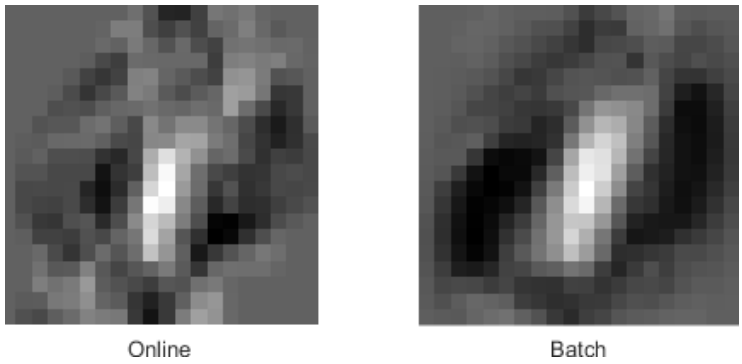

Figure 5: Model weights after training of a *pixelwise* model. The 1D weigh vectors were reshaped into a 2D image, where each pixel shows how their activation affects the resulting classification: dark regions increase the confidence of the image being of a zero; white regions, of being of a one

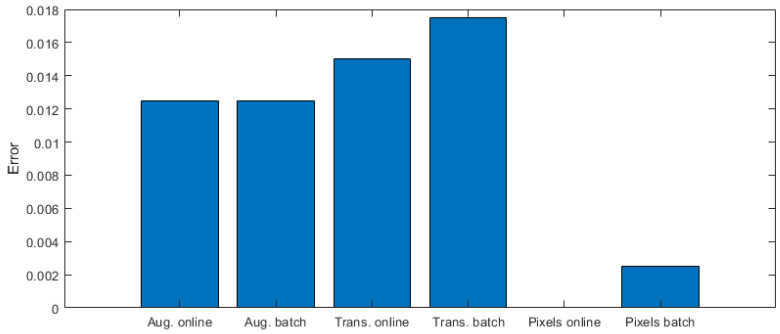

Figure 6: Error rates of all six models: the *online* and *batch* variants of the *augmented*, *transformed* and *pixelwise* models

worse results than the *augmented* ones; a surprising result, considering that it performed better against the training set. This decrease in performance could be due to overfitting. The *pixelwise* models performed the best among all, with the *online* version having zero misclassifications.

2 Linear Basis Function Models for Regression

2.1 The online LMS learning rule

The resulting optimal weight vector $\mathbf{w}^*$ is always different depending on the randomly generated training samples and the number of training iterations. Using the training samples shown in Figure 7 as blue circles result in a $\mathbf{w}^* =$

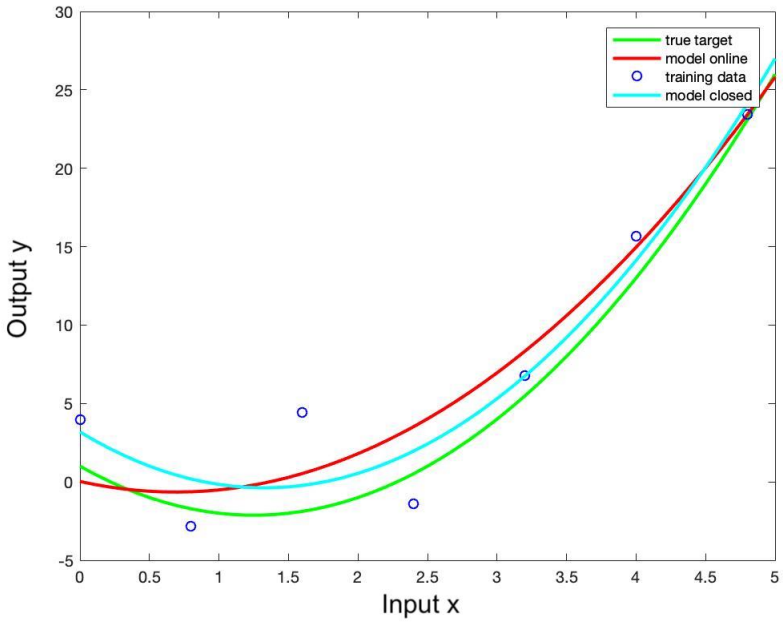

Figure 7: Results of the LMS learning rule in an online and closed version

$[1.42, -1.96, 0.03]$. This weight vector was obtained after 811 training iterations and causes an error of $e = 62.71$. The number of iterations differs strongly on the randomly generated training samples. We terminate the training if one of these three conditions is fulfilled:

- a maximum number of iterations is reached
- the error drops below a threshold
- the error converges (difference of errors in 2 subsequent iterations drops below a threshold)

In this case, the error converged.

2.2 Comparison of online and closed version

We obtain the closed form of $\mathbf{w}^*$ with

$$\mathbf{w}^* = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{t}^T, \tag{1}$$

whereas we use Moore-Penrose inverse to compute an inverse matrix. We obtain $\mathbf{w}^* = [2.03, -5.38, 3.19]$. Computing $\mathbf{w}^*$ in closed form requires only one iteration. To compare the online version and the closed version we performed

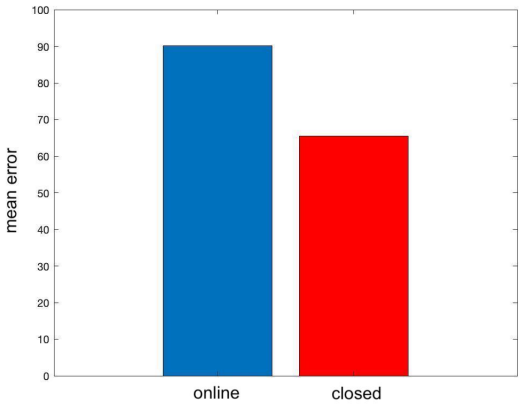

Figure 8: Comparison of the mean errors for the online and the closed version

an experiment. We computed the mean error of $\mathbf{w}^*$ on the training set, both for the online version and the closed version. We obtained $\mathbf{w}^*$ for 11 different training datasets, both in the online and closed version. By comparing the mean errors of the online and closed version, shown in Figure 8, we can observe that the closed version produces a lower mean error than the online version. Hence, the closed version gives us more accurate results. However, it can get expensive to compute the pseudoinverse for bigger datasets. Therefore we recommend using the closed version for small sets of training data and the online version for bigger sets of training data.

2.3 The learning rate γ

The learning rate γ defines how fast the weight vector w adapts during the training process. When using a high learning rate the weight vector gets adapted relatively fast, but it is more likely that the local minima of the error can not be reached because the error is more likely to diverge. Using a small learning rate will result in more iterations, a slower adaption of the weight vector, but a finer search of the local minimum. That is it is necessary to make a tradeoff between the number of iterations and convergence. Because of that, it is beneficial to use a time-varying learning rate $\gamma(t)$, where gamma is dependent on the number of iterations of the training algorithm. At the beginning of the training, it is helpful to use a higher learning rate to have a quick adaption of the weight vector. At this point it is sufficient to make bigger steps because we are further away from a minimum. While later in the training process, when we are closer to the minimum, the adaptions of the weight vector should be smaller to not 'overshoot' the minimum. Figure 9 shows two versions of a time dependent learning rate $\gamma(t)$. In our code we use the learning rate $\gamma(t) = \frac{0.025}{\sqrt{t}}$, which is defined by the blue curve. The brown curve shows the learning rate $\gamma(t) = \frac{0.025}{t}$,

which is proposed in the lecture slides. However, we the error converges faster using the $\gamma(t) = \frac{0.025}{\sqrt{t}}$. The brown curve converges faster to 0 than the blue curve. Values that are very close to 0 can cause numerical issues because they can not be represented by floating-point variables. $\gamma(t) = \frac{0.025}{\sqrt{t}}$ does not have this problem which is also a reason why we have chosen it as our learning rate function.

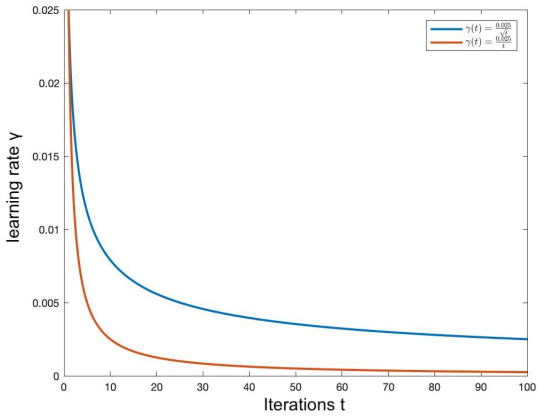

Figure 9: two versions of a time varying learning rate $\gamma(t)$

3 Linear basis function model with image data

In Figure 10 you can see the 7 generated images used for calculation of estimation of weight vector $\mathbf{w}^*$ in closed form. This is the training set. Each figure from training set was transformed into a vertical vector. A numeric constant $x_0 = 1$ was augmented to each vector at the top. All the images were this way transformed into a matrix of $29^2 + 1 = 842$ rows and 7 columns. That is our input matrix $\mathbf{X}$. The target values $\mathbf{t}$ were calculated similarly as in previous section, skipping the additive noise.

We would like to use equation 1 to calculate $\mathbf{w*}$ just as in previous section. Unfortunately, we cannot directly compute inverse $(\mathbf{X}\mathbf{X}^T)^{-1}$ as the matrix $(\mathbf{X}\mathbf{X}^T)$ is singular. This may be caused by the fact, that borders of most of train images consist purely of zero sequences which results in linearly dependent rows and therefore singular matrix $(\mathbf{X}\mathbf{X}^T)$.

To regularize the matrix $(\mathbf{X}\mathbf{X}^T)$ we used ridge regression method. Basically, we add a (ridge) penalty to coefficients that cause weight vector estimate $\mathbf{w*}$ values to be large. We do so by substituting $(\mathbf{X}\mathbf{X}^T)$ by $(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})$. The parameter λ determines the penalty rate. In further calculations we used value $\lambda = 0.1$ which is approximately one order below the maximal value of $(\mathbf{X}\mathbf{X}^T)$ matrix elements. Experimenting with different values didn't lead to significantly

Figure 10: Training sample images.

different results of **w*** and estimation errors. Namely the influence is inferior to the influence of circle center coordinates noise..

Having the inverse we were now able to calculate weight vector estimation **w*** and the predicted values $\hat{y}$ for training images, which can be seen in Figure 11. The estimates are visually very close to true targets. The resulting residual sum of squares (RSS) error of the training images prediction is equal to

$$E(\mathbf{w}*) = \sum_{i \in trainSet} (t_i - \hat{y}_i)^2 = 0.01107[-]. \tag{2}$$

In Figure 12 you can see a similar pattern, this time displaying the predictions $\hat{y}$ of all 51 generated images, therefore now mostly the ones weight vector **w*** is not based upon. Here we can see that the predictions are scattered around the true target with significant errors, but the tendencies of prediction and true target are similar.

If we increase variance of circle center coordinates in input images, the error of predictions should also be increased as the similarity of train images and the rest images gets lower. This pattern can be seen in Figure 13. The x-axis variance starts at value 4 (corresponds to previous calculations) and increases to the right. On the y-axis, the RSS prediction error seems to be also getting larger. As each variance value in the graph is represented by one set of 51 images, the values are significantly influenced by randomness. To get an even more valid pattern we could consider multiple sets of images for each variance value and calculate mean error for each set of sets.

In Figure 14 you can see the weight vector estimate **w*** visualized in a 2-D image form. For sufficient visibility, the weight values were linearly normalized to range $\langle 0, 1 \rangle$. Comparing Figures 14 and 10 we can see the influence of each train images on resulting **w***, which visually resembles the combination of overlaying train images.

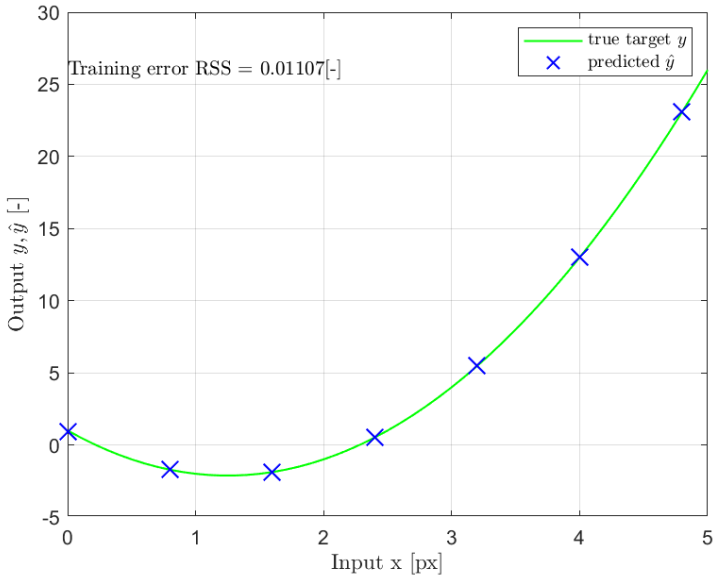

Figure 11: Comparison of true target y to train sample images $\hat{y}$ prediction.

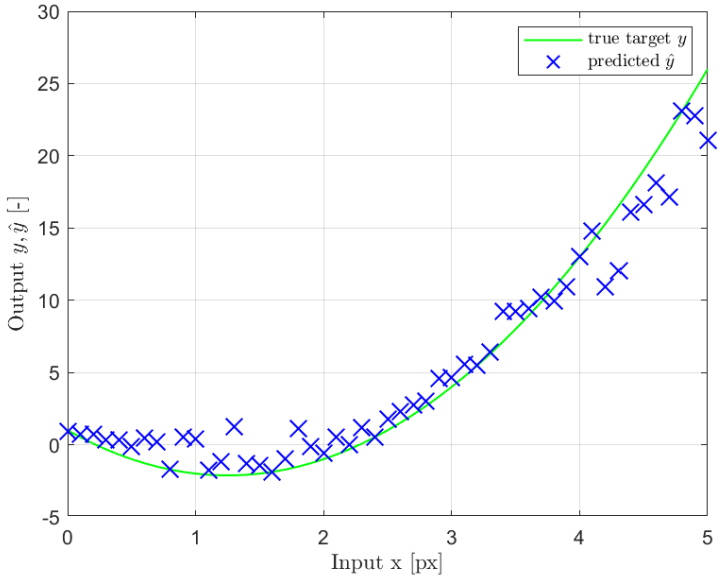

Figure 12: Comparison of true target y to full sample images $\hat{y}$ prediction.

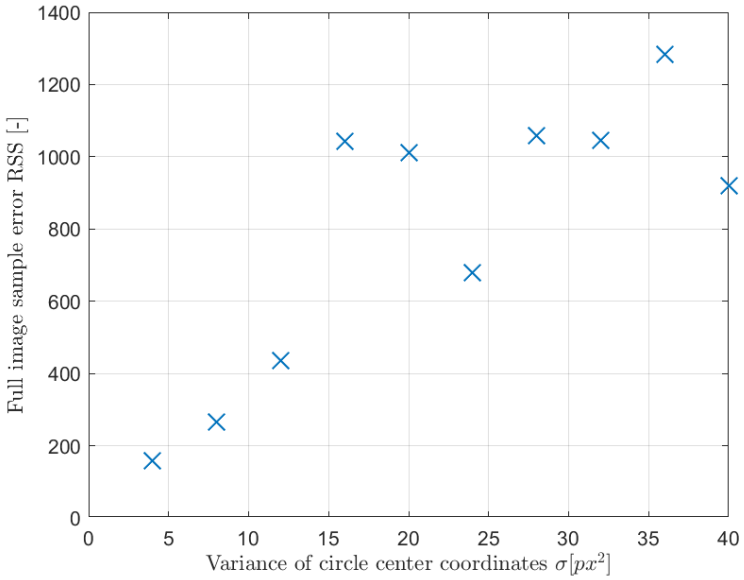

Figure 13: Influence of circle coordinates variance to RSS of full image sample.

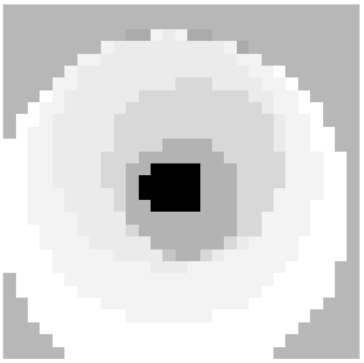

Figure 14: Weight vector estimate $w*$ image representation. Weight values are linearly normalized to range $\langle 0, 1\rangle \mathrel{\widehat{=}} \langle black, white\rangle$.