# 183.605
# Machine Learning for Visual Computing
# Assignment 1

Patrick Link
Arnaud Nativel
Fabian Pechstein

28. Oktober 2018

## 1 Task1

### 1.1 MNIST Data Set and Feature Selection

We applied a perceptron model to classification of handwritten digits. MNIST is a standard data set with $28 \times 28$ pixel images of handwritten digits. We selected the digits $0$ and $7$ and 500 training images for each class for our classification task, as they are reasonably different. To chose the right feature we plotted all regionprops in a scatter plot matrix,see figure 1 and selected solidity and eccentricity. To compare the batch and online version we used an equivalent number of maximal iterations. This means we took used the size of a batch times the maximum number of iterations in the batch case as the maximum number of iterations for the online algorithm. In our case we took $500000$ and $500$.

#### 1.1.1 Features without transform

We used once the batch training algorithm and once the online training algorithm to classify the data directly from these to features. The connfusion matrix evaluated on a separate test set gives us following perfomance:

For the online training algorithm evaluated on the test set we got 385 correctly classified results of 400 test images. We got the following confusion matrix:

| n=400 | classified as 1 | classified as 7 |
|---|---|---|
| Digit 0 | 185 | 15 |
| Digit 7 | 0 | 200 |

For the batch training algorithm evaluated on the same test set we got 383 correctly classified results of 400 test images. We got the following confusion matrix:

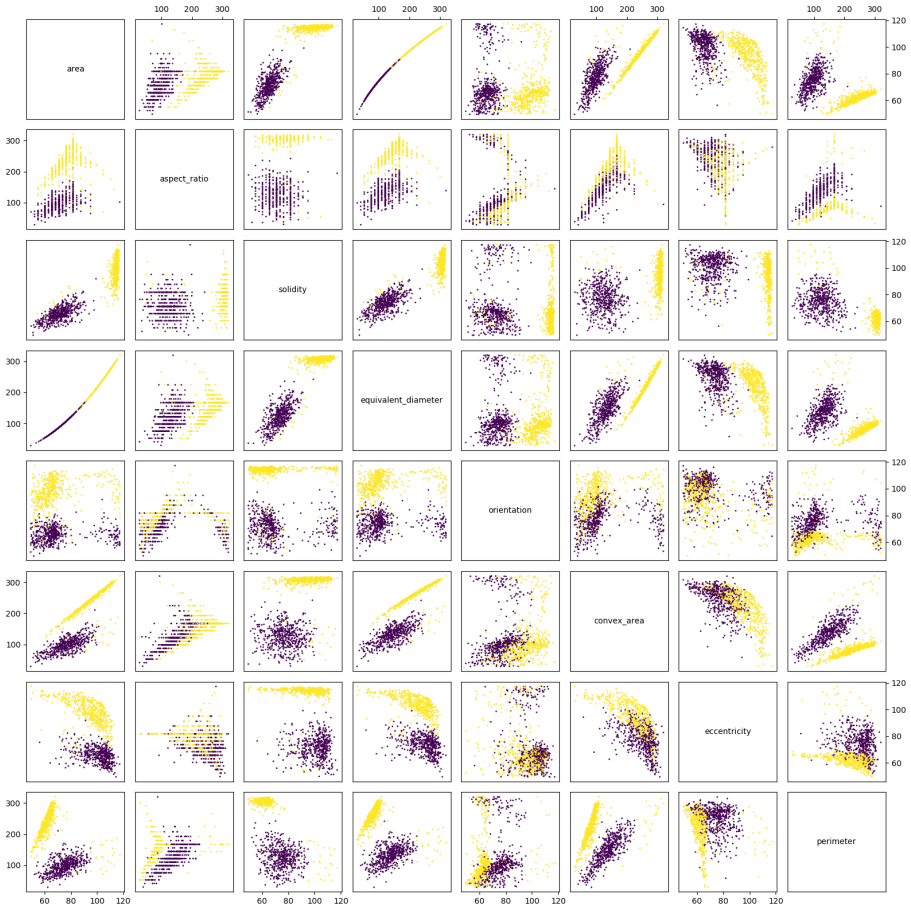



Figure 1: Scatter plot matrix of all region properties

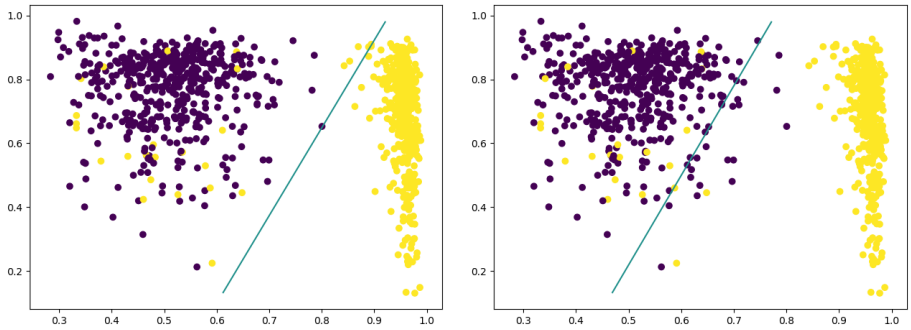


Figure 2: The decision boundry of the trained perceptron. On the left for the online algorithm on the right for the batch algorithm.

| n=400 | classified as 1 | classified as 7 |
|---|---|---|
| Digit 0 | 187 | 13 |
| Digit 7 | 4 | 200 |

The corresponding decision boundries are shown in figure 2. We see that the data is not linearly separable, because there are yellow points inside the purple "cloud".

In theory a perceptron is able to detect a linearly separable set, because if the set is linearly separable, then the algorithm is guaranteed to converge. Moreover we can bound the number of training steps from above. But as this bound depends on a separating hyperplane and the training set, we can not know beforehand how big this upper bound is.

### 1.1.2 Classification with feature transform

We performed the same experiments as above, but first applied following feature transform to our data. $(x, y) \mapsto (1, x, y, x^2, y^2, xy)$

For the online training algorithm evaluated on the test set we got 386 correctly classified results of 400 test images. We got the following confusion matrix:

| n=400 | classified as 1 | classified as 7 |
|---|---|---|
| Digit 0 | 186 | 14 |
| Digit 7 | 0 | 200 |

For the batch training algorithm evaluated on the same test set we got 383 correctly classified results of 400 test images. We got the following confusion matrix:

| n=400 | classified as 1 | classified as 7 |
|---|---|---|
| Digit 0 | 187 | 13 |
| Digit 7 | 4 | 200 |

The corresponding decision boundaries are shown in figure 4. It's not very surprising that the feature transform doesn't help the classification. as there are yellow points, corresponding to the digit 0, in between purple points. Such a simple feature transform can't separate those points, as the new decision boundary is just a quadratic.

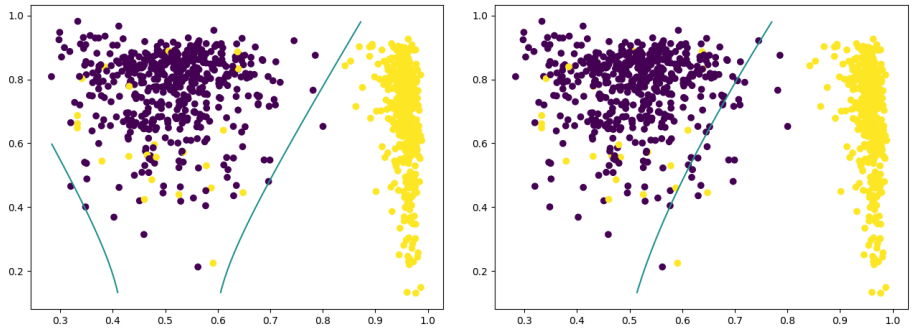



Figure 3: The decision boundry of the trained perceptron. On the left for the online algorithm on the right for the batch algorithm.

### 1.1.3 Classification directly with image data

We then tested the perceptron algorithm by using the image data as direct input. This is done by vectorizing the image into one vector. Instead of showing the corresponding decision boundary, we will visualize the calculated weight vector of the perceptron this is shown in figure **??** for both the online and the batch algorithm.

We see that the weights resemble a bright 0 overlayed over a dark 7. This follows from the fact, that the perceptron tries to give images resembling a 7 a positive value and images resembling a 0 a negative value. We can also see that this is much clearer in the batch algorithm. We think this follows from the fact, that in the online algorithm the last data point observed has a big influence on the resulting weight vectors. On the other hand, as we compare a weight vector over the whole data set in the batch algorithm, each singular data point has much less relative influence on the resulting weights.

Moreover, in this experiment the training set was linearly separable. And we got very good results on the test set, where the online algorithm even classified all images correctly.

For the online training algorithm evaluated on the test set we got 400 correctly classified results of 400 test images. We got the following confusion matrix:

| n=400 | classified as 1 | classified as 7 |
|---|---|---|
| Digit 0 | 200 | 0 |
| Digit 7 | 0 | 200 |

For the batch training algorithm evaluated on the same test set we got 398 correctly classified results of 400 test images. We got the following confusion matrix:

| n=400 | classified as 1 | classified as 7 |
|---|---|---|
| Digit 0 | 200 | 0 |
| Digit 7 | 2 | 198 |

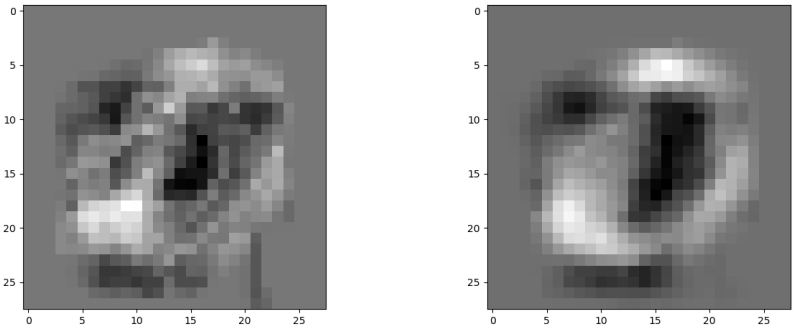


Figure 4: On the decision boundary of the trained perceptron. On the left for the online algorithm on the right for the batch algorithm.

## 1.2 Performance

In the following table we summarize the performance of each experiment. We only report the error rate, i.e. ratio of wrongly classified images, because the confusion matrices are already given above. We evaluated the perceptrons on a test set with 200 images of each class.

| Experiment | Online error rate | Batch error rate |
|---|---|---|
| 2d features | 0.0375 | 0.0425 |
| 5d features | 0.035 | 0.0425 |
| whole Images | 0 | 0.005 |

# 2 Task2

## 2.1 LMS vs. closed Form

We followed the instructions and used the function below with $G = 6$ to determine our $y$ values for the experimental setup.

$$f(x) = 2x^2 - Gx + 1 \tag{1}$$

### 2.1.1 What is the resulting weight Vector when using the LMS rule?

We implemented an online LMS learning rule for regression to determine useful weights to fit the curve. Figure 6 shows four runs with randomised training data. As we can see the fitted curve is moved towards the original curve with each iteration.

### 2.1.2 How can you determine the optimal $w*$ in closed form? Compare $w*$ with the outcome of the LMS-rule training

We used Moore-Penrose-Inverse to minify the sum of squares error. As displayed the in the Tables, 1 and 2 we can see the resulting weights and the calculated errors.

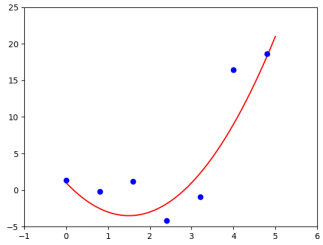


Figure 5: Setup and an example of $t_i$ values with additional noise

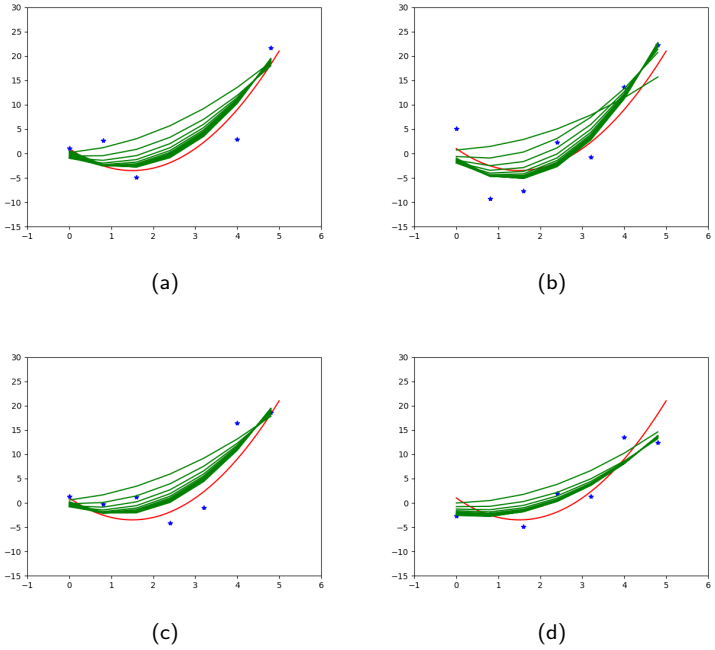
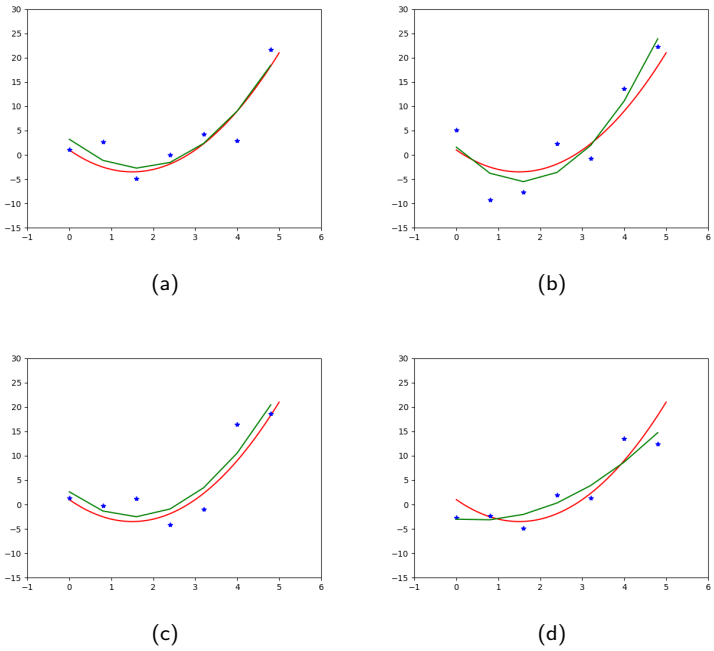


Figure 6: Visualization of the learning process of **LMS**

We can see that the error rate for our four examples with CF are all lower compared to LMS, without much surprise.

### 2.1.3 Influence of $\gamma$

$\gamma$ has influence over the fact if the algorithm can converge in the end. In case $gamma$ is set too high, new values get too much weight, thus 'overriding' older



(a) (b) (c) (d)

Figure 7: the same training data as in figure, however this time solved in closed form (**CF**)

| **Figure** | **LMS** | **CF** |
|---|---|---|
| a | 97.09004286828198 | 71.03243449830224 |
| b | 112.43543529868002 | 97.82006516766448 |
| c | 109.4839763954856 | 84.46023189949588 |
| d | 61.164485036889054 | 47.10797373501289 |

Table 1: error comparisons betweenLMS and CF

ones and introducing change to quickly. See Figure 8 for an example.

| Figure | LMS | CF |
|---|---|---|
| a | 2.79641366 -4.06370085 1.6057828 | 6.10503808 -6.93083294 2.04038233 |
| b | 0.90857717 -6.42409915 2.36363687 | 1.57854501 -8.9662967 2.8358482 |
| c | 0.25694124 -4.13859362 1.69469808 | 2.59983553 -6.64808401 2.1588638 |
| d | -2.58665078 -0.97237072 0.9121616 | -3.01226058 -0.92406774 0.96261322 |

Table 2: resulting weightsLMS and CF

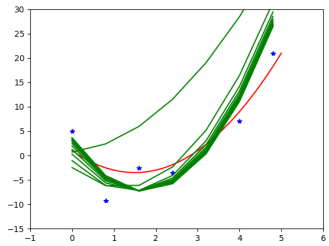


Figure 8: $\gamma$ has been chosen too high

## 2.2 Image Data

In this part, we kept the same data set and target values than the previous experiment, but we decided to take another representation of the input data $X$. Instead each $x$ will be represented by a $29 \times 29$ matrix (an image of a circle) where each entry $(i, j)$ is equal to $1$ if $(i - m_1)^2 + (j - m_2)^2 - (3x)^2\ 0$ and $0$ otherwise. Another important point is the addition of a noise not on the target value but on the features of the training set, with the 2 center's coordinates$(m1, m2)$ following a normal distribution $\mathcal{N}(15, 2)$.

The first part of the experiment was to compute the best fitting weights in closed form, by using the pseudo-inverse of the matrix of the training set, which we previously homogenized. We vectorized the transform $\Phi X$ and set the matrix $X_ij = \Phi_j(x_i)$. We then obtained a 842-array representing weights. With those weights, we computed the error between the target values and the predicted values for the trainin set and we obtained an error of 0. Then, we compared the target values and the predicted values for the whole data set. The results are shown in figure 9.

To see the importance of the noise, we did the experiment several times with varying the variance of the noise.

It appeared that the bigger the variance was, the farer the predicted values were from the target values. One remarkable fact is that the 2 curves don't even cross at the training data x value, which can be explained by the fact that when testing the weights on the full data set, we also recompute the images of the training set.

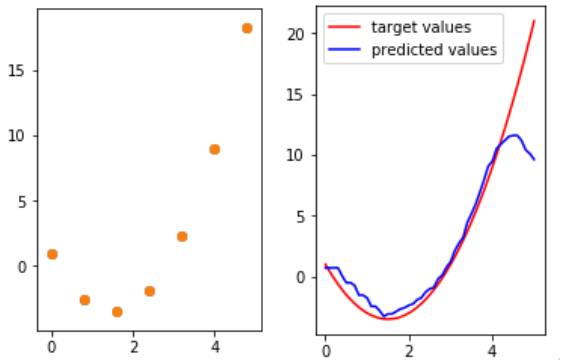



Figure 9: Predicted values on the training set and comparison on whole range

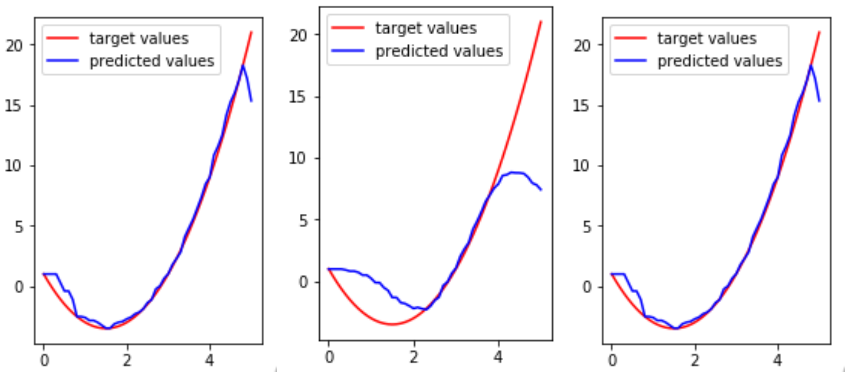



Figure 10: Influence of the variance on the error of the regression. Variance of 0, 4 and 9 is used from left to right.