MLVC Assignment 2 Report

Jakob Troidl Lucas da Cunha Melo Martin Ptacek

January 2020

1 The dual optimization problem

As training dataset, we use two features extracted from images of the MNIST set (containing handwritten digits from 0 to 9), namely *filled area* and *solidity*. First, we load a set of $N = 400$ images of only zeros and ones (200 of each) and extract the features. We then make sure the data is linearly separable by manually removing selected points, resulting in $N = 394$ points (197 of each class).

1.1 Training SVM

We implemented the function

```
[alpha, w0] = trainSVM(X,t,C,kernel)
```

with the following parameters:

- $\mathbf{X}$ is a the input data; a $N \times d$ matrix with N train samples with d dimensions. For the experiments in this section, $N = 394$ and $d = 2$.
- $\mathbf{t}$ is a $N \times 1$ vector of train data labels with value 1 or -1.
- C is a penalty strength in connection to slack variables (described in Section 2).
- `kernel` is a kernel function handle (described in Section 2).

To solve the optimization problem for the `alpha` values, we use the Matlab function

```
[alpha] = quadprog(H,f,A,b,Aeq,beq)
```

with the following parameter setting:

- $\mathbf{H} = (\mathbf{X} \cdot \mathbf{t}) \cdot (\mathbf{X} \cdot \mathbf{t})^T$
- $\mathbf{f} = -\mathbf{1}_{N\times 1}$
- $\mathbf{A} = -\mathbf{I}_{N\times N}$

- $\mathbf{b} = \mathbf{0}_{N\times 1}$
- $\mathbf{A}_{eq} = \mathbf{t}^T$
- $b_{eq} = 0$

which finds the $N \times 1$ matrix $\boldsymbol{\alpha}$ that solves

$$min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}'\mathbf{H}\boldsymbol{\alpha} + \mathbf{f}'\boldsymbol{\alpha}$$

over the constraints

$$\mathbf{A} \cdot \boldsymbol{\alpha} \leq \mathbf{b}$$

$$\mathbf{A}_{eq} \cdot \boldsymbol{\alpha} = \mathbf{b}_{eq}$$

Using the optimized $\boldsymbol{\alpha}$ values, we are able to directly calculate w_0 vector. Together they define the decision boundary, enabling the classification of future points. This boundary, its margins and the support vectors are visualized together with the data, shown in Figures 1 and 2. We then implemented the function

```
y = discriminant(alpha, w0, X, t, Xnew);
```

which calculates the discriminant of new data points (`Xnew`) using the trained SVM ($\boldsymbol{\alpha}$ and w_0). The signum of the output `y` vector gives us the classification of `Xnew`.

1.2 Testing SVM

In Figure 3 we can see a trained SVM (boundary) used to classify unknown test data set (red $\times$ and green $+$ symbols). The classification of the test set did not result in any error, and therefore the chosen test set of $N = 400$ samples is also linearly separable.

2 The kernel trick

2.1 Radial basis function kernel

We chose the Radial Basis Function as a kernel. To implement it for two 2D input vectors $\mathbf{x_{1,i}}$ and $\mathbf{x_{2,j}}$, each representing 1 data sample properties filled area and solidity, we write the equation

$$K(\mathbf{x_{1,i}}, \mathbf{x_{2,j}}) = e^{\frac{-|\mathbf{x_{1,i}} - \mathbf{x_{2,j}}|^2}{\sigma^2}}.$$

We implemented a Matlab function `rbfkernel()` that inputs multiple $\mathbf{x_{1,i}}$ and $\mathbf{x_{2,j}}$ vectors (data samples) in matrices $\mathbf{x1}$ and $\mathbf{x2}$ and calculates $k_{i,j}$ for each vector combination, creating an output matrix $\mathbf{k}$ of $k_{i,j}$ coefficients of same dimensions as matrix $\mathbf{x1} \cdot \mathbf{x2}$.

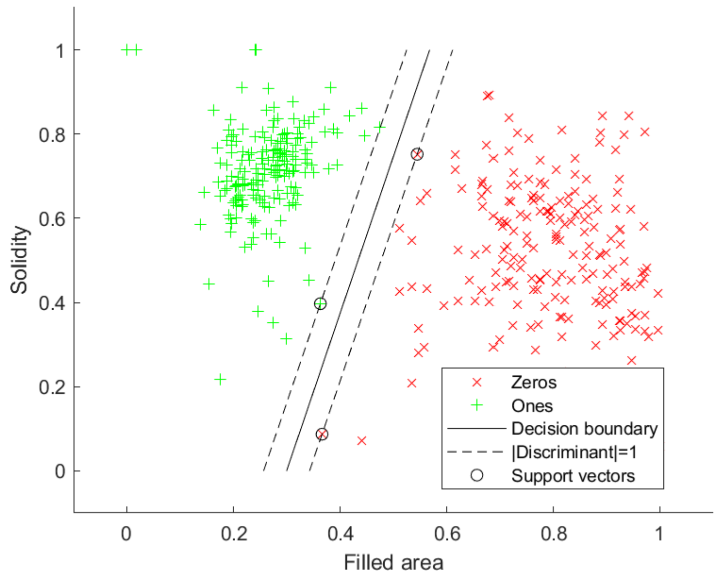

Figure 1: Trained SVM.

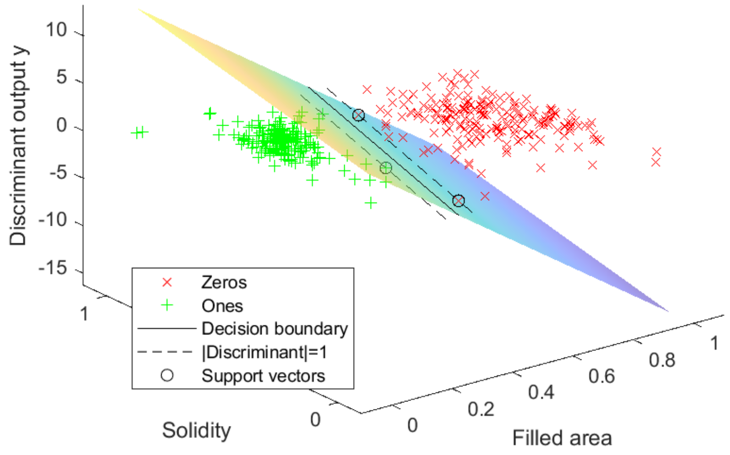

Figure 2: Trained SVM with discriminant function surface.

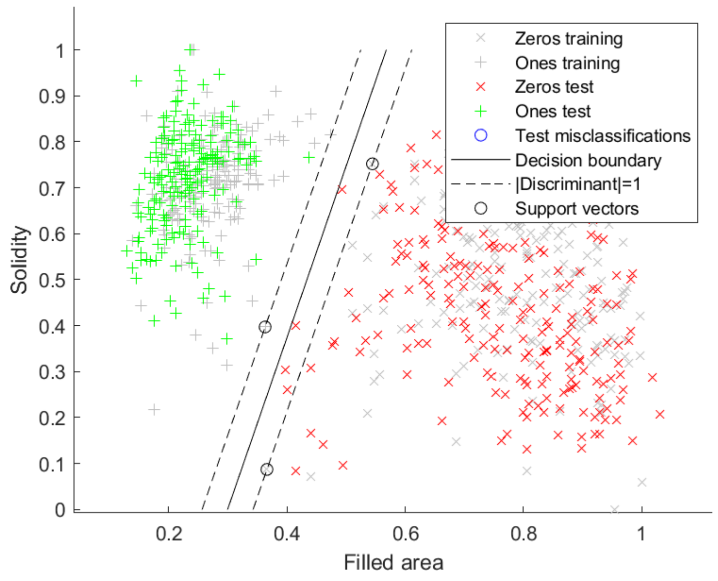

Figure 3: Testing SVM on another $N = 400$ long set.

```
function [k] = rbfkernel(x1, x2, sigma)
```

Later, the `rbfkernel()` function was modified to support an arbitrary number of dimensions for the input, as this is needed for the experiments in Section 3.

The `rbfkernel()` function is then passed as a handle to the `trainSVM()` and `discriminant()` functions. To enable that, we must pass the `sigma` parameter as well. Passing it as an additional argument would force us to rewrite the function's argument lists. Moreover, other kernels may need different parameters (e.g. linear kernel does not use any additional parameters). Therefore, we decided to pass the additional parameter using a specific function handle as depicted below:

```
1    sigma = 0.2;
2    kernelFunc = @(x1, x2)rbfkernel(x1, x2, sigma);
3    [alpha, w0] = trainSVM(X, t, kernelFunc, C);
```

In the second line, we crate a handle with the determined sigma parameter. Therefore, the function available through the handle keeps only 2 arguments. This handle is passed to `trainSVM()` as in Line 3 and later to `discriminant()` as well.

In the `discriminant()` function, the modification is replacing $\mathbf{X} \cdot \mathbf{X}_{new}{}^T$ with `kernel(X, Xnew)`. In the `trainSVM()` function, we use the kernel to change the way matrix H is calculated for the `quadprog()` optimization:

- $\mathbf{H} = kernel(\mathbf{X}, \mathbf{X}) \cdot (\mathbf{t} \cdot \mathbf{t}^T)$.

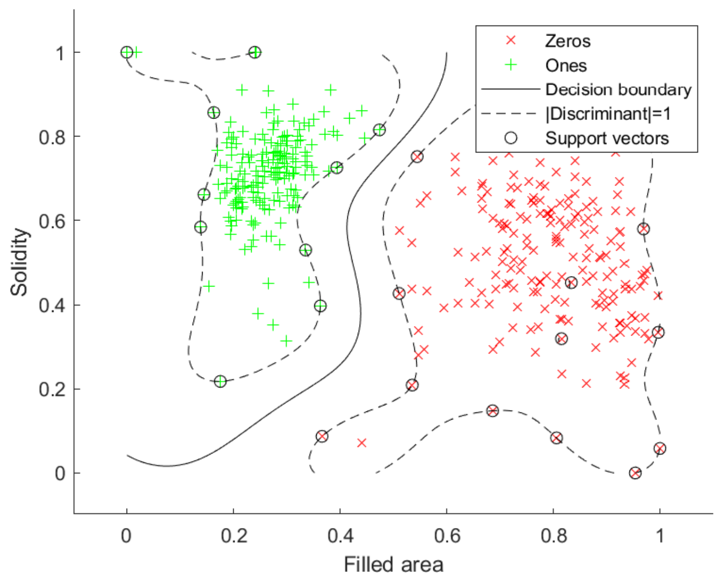

Figure 4: Decision boundary for $\sigma = 0.2$ and $C = Inf$.

The calculation of w_0 must also use the kernel in order for the support vectors to get a discriminant with value ± 1.

In Figure 4 we show the changed decision boundary for the same linearly separable data, this time using a radial basis function kernel. It is apparent that, using a sufficiently low σ parameter, this configuration is able to cope with a non-linearly separable data set.

In Figure 5, we show the surface of the discriminant function for the whole parameter space of the 2-dimensional input. It uses the same value of the parameter σ; therefore the decision and support regions are the same as in Figure 4.

2.2 Influence of σ

Figure 6 depicts the influence of RBF parameter σ on the shape of decision boundary. We use a logarithmic range $[10^{-2}, 10^{1}]$. With higher values of σ, the decision boundary gets straighter and more similar to a linear kernel behavior. Decreasing σ leads to a more curved boundary, which then, when getting too low, encircles individual data points. That is a symptom of overfitting.

2.3 Slack variables - regularization parameter C

The regularization parameter C sets an upper bound for the α values. When applying a certain C, we get $\alpha \in [0, C]$. That is why we also get $|d(x_s)| \leq 1$,

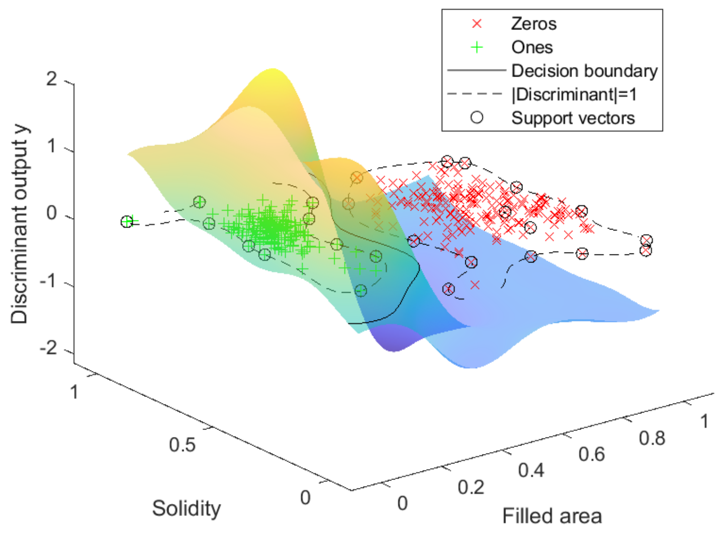

Figure 5: Discriminant function surface for $\sigma = 0.2$ and $C = Inf$.

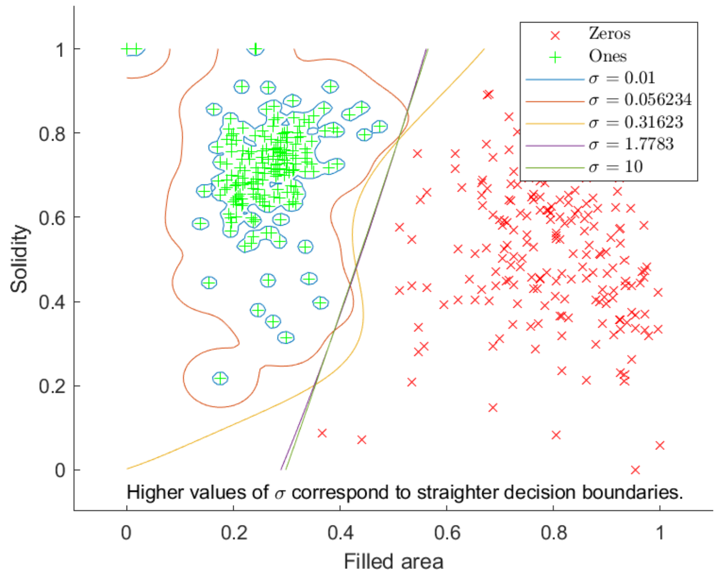

Figure 6: Decision boundary for various σ values and $C = \infty$.

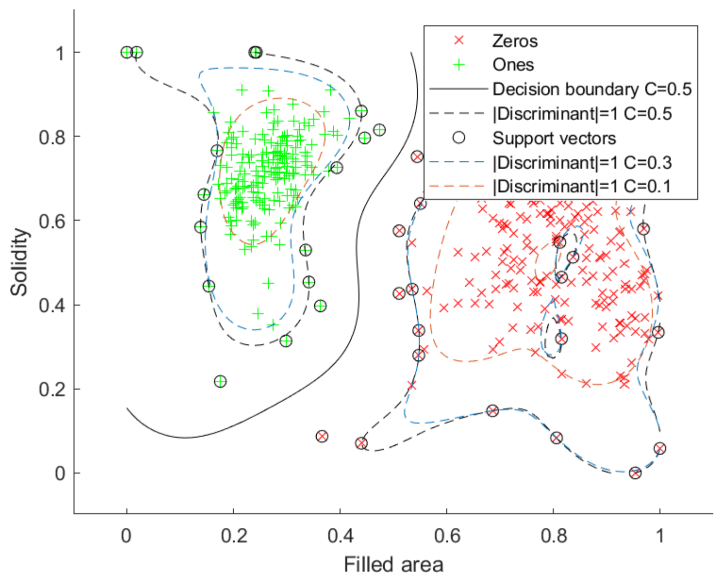

Figure 7: Decision boundary for $\sigma = 0.2$ and various C.

where $d(x_s)$ is the discriminant value of an arbitrary support vector. In other words, that means that we allow data samples to lie inside the margin region. Having a $C < \infty$ also allows misclassified data samples during training.

When computing w_0, we want to make sure that we use a x_s with margin of $|d(x_s)| = 1$. By taking a x_s with a corresponding α in the range of $0 \ll \alpha \ll C$, we make sure to fulfill this condition.

Figure 7 shows various configurations of C. A smaller value of C causes the margin lines to be shrunk closer around the data samples.

2.4 Testing on data set of *assignment1*

In this section, we considered the training data set of *assignment1*, which consists of 400 2-dimensional samples and is linearly non-separable. Using parameter values of radial basis kernel σ and penalty strength C to observe the effect of curved decision boundary and accepting sample errors, we obtained a visually understandable decision boundary in Figure 8.

3 Model-complexity and model-selection

In this section, we evaluate our implementation of the Support Vector Machine. First, we compare a linear SVM (linear kernel function, no slack variables) with our implementation of the perceptron.

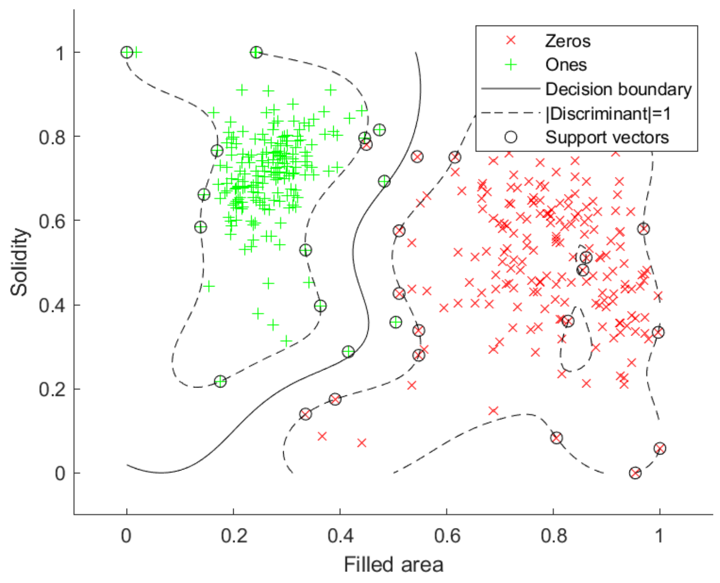

Figure 8: Decision boundary and test data for $\sigma = 0.2$ and $C = 2$.

3.1 Comparison of the Perceptron and a linear SVM

We train 150 SVMs and 150 perceptrons using 150 distinct training sets of 70 images of 0 and 1 digits of the MNIST dataset. We test each of the SVMs and the perceptrons using all test images of 0 and 1 digits of the MNIST dataset. For each SVM and perceptron, we compute the proportion of misclassified test images. By averaging this value over all 150 versions of the SVMs and perceptron, we get the average error R_{avg}.

Figure 9 shows R_{avg} for the perceptron and the linear SVM in %. Due to the high dimensionality, the probability is high that the test set is linearly separable. That is why as well the perceptron as the linear SVM have a relatively low R_{avg}. However, the linear SVM performs better than the perceptron because it finds a decision boundary that maximizes the margin region.

3.2 Tuning the meta-parameters C and σ when using an RBF-kernel

We performed our evaluation of R_{avg} over a set of values over the 2D parameter space which is defined by (σ, C). Figure 10 (left) shows R_{avg} for different parameter configurations. Low values of σ produce a high R_{avg} value. In that case, the SVM overfits on the sparse training data, which does not generalize well on the big test data set. In general, higher C values produce also a lower R_{avg}.

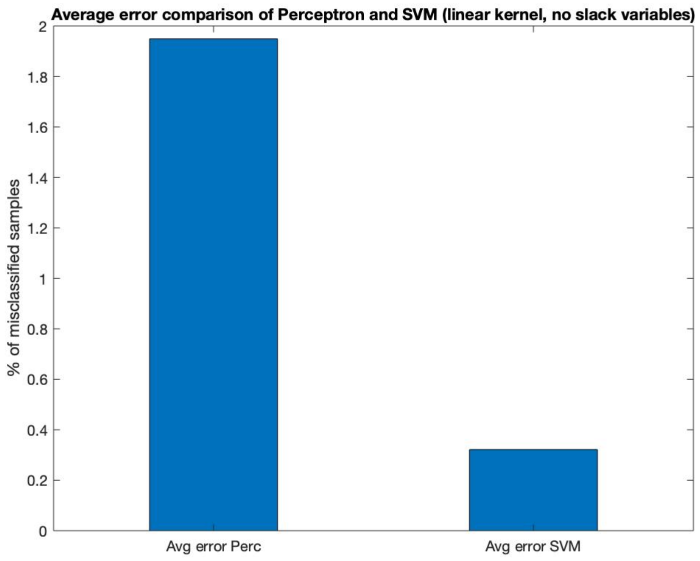

Figure 9: Comparison of the average error R_{avg} in % between the perceptron and a linear SVM

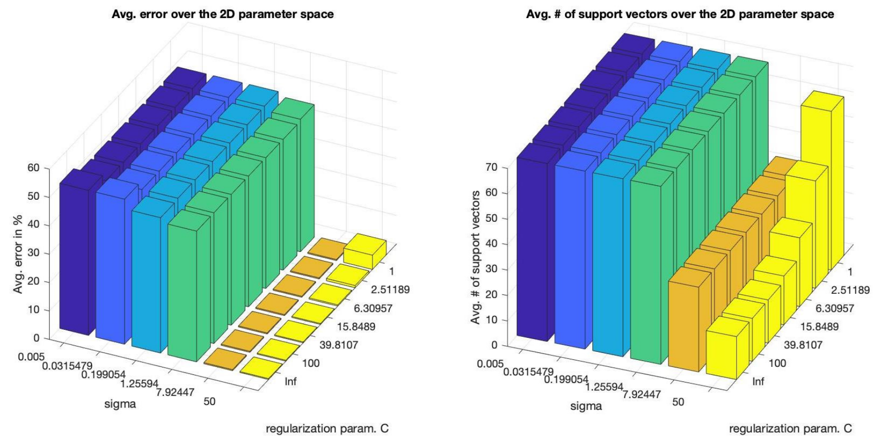

Figure 10: (left) average error of the SVM with different configurations of σ and C. (right) average number of support vectors over different configurations of σ and C.

Figure 10(right) shows the average number of support vectors of the 2D parameter space. There is a close correlation between the number of support vectors and the R_{avg}. Parameter configurations with a high number of support vectors have also a high R_{avg}, while a low number of support vectors corresponds to a more general model and therefore is more likely to have a small R_{avg}.

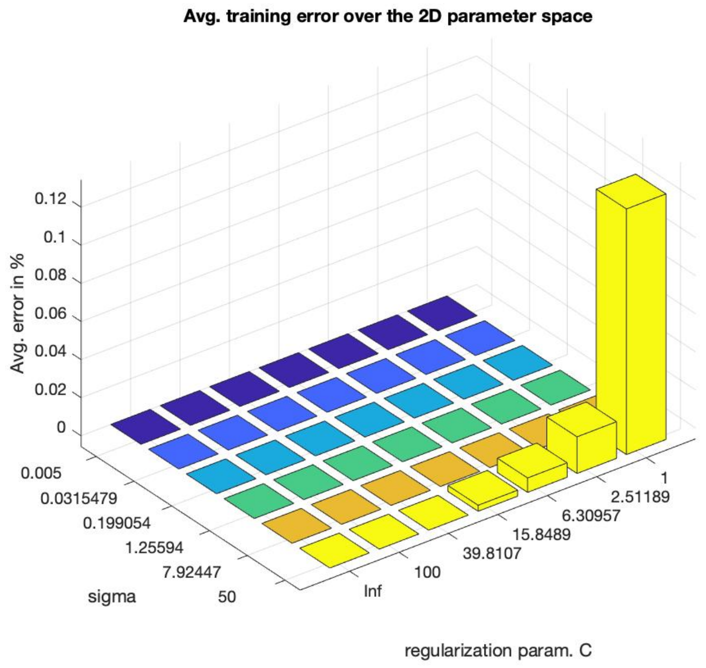

Figure 11: Average error when testing against the training set with different parameter configurations.

Figure 11 shows R_{avg} when using the training set also as a test set. As one would expect $R_{avg} = 0$ for most parameter configurations. However, we observe an $R_{avg} \sim 0.1\%$ for $C = 1$ and $\sigma = 50$. As we have shown in section 2.3, high values of σ correspond to more straight decision lines. Small C values allow more misclassifications during training. An approximately linear decision boundary cannot overfit to the data. When having a high value of C, misclassifications during training happen, which are visible in that case.

3.3 Optimization using cross validation

Again using $M = 150$ distinct sets of 20×20 MNIST dataset images, flattening the them into 400×1 matrices, and using these matrices as input in the SVM, we now aim to find optimal C and σ parameters by M-fold cross validation. For initial experiments, we populated each of the M sets with $N = 20$ flattened images and used differing C and σ values. The results are shown in Figure 12.

Based on these results, we selected $\sigma = 10$ and $C = 25$ as optimal parameters, as this combination is the one with lowest average error (0.86085%).

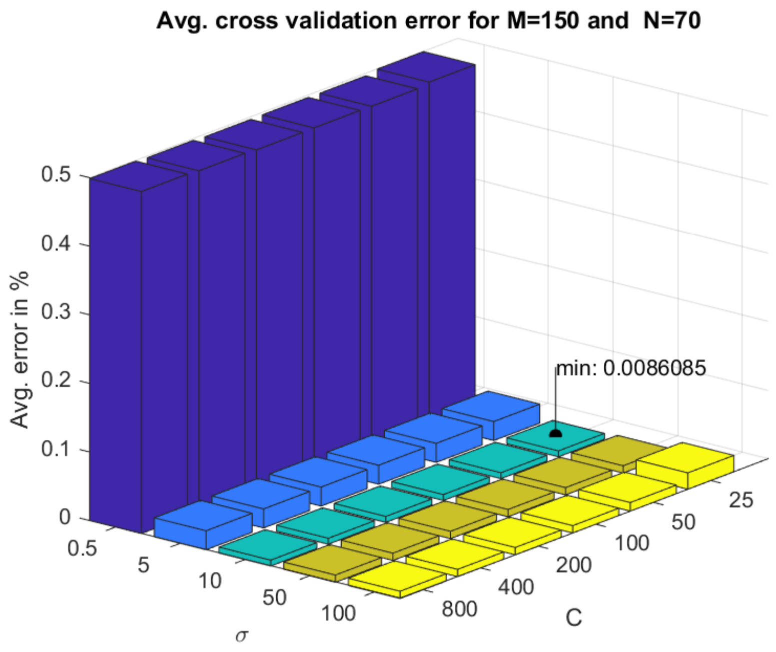

Figure 12: Average cross validation error with different C and σ combinations. The combination with smallest error is highlighted.

Having found the optimal parameters $\sigma = 10$ and $C = 25$, we then compare a non-linear SVM using these parameters with a linear SVM (no kernel, no slack variables). The comparison is made by training $M = 150$ SVMs of each type, each using $N = 70$ flattened MNIST images as input, and testing the resulting SVM using the MNIST test set.

The error of each of the $M = 150$ SVMs is then averaged and the result is plotted in Figure 13. In this case, the RBF-kernel SVM (non-linear) outperforms the linear SVM by a narrow margin.

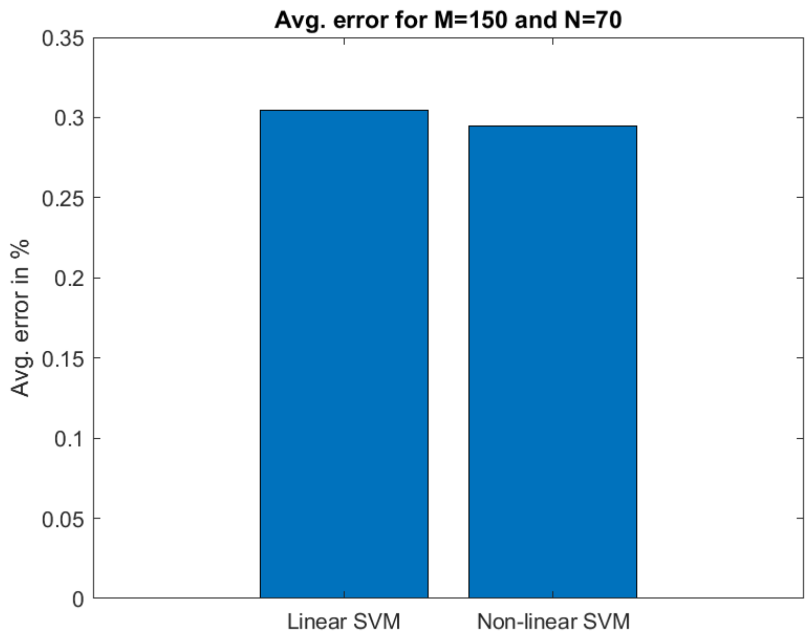

Figure 13: Average error of $M = 150$ linear and non-linear (using an RBF-kernel) SVMs. Each SVM uses $N = 70$ flattened MNIST training images for training. The classification is performed on the MNIST test set. Linear SVM average error: 0.30442%. Non-linear SVM average error: 0.29456%.