universität wien

Training SVM- and kNN-Classifiers on a reduced MNIST data set

Seminar Optimization

Author
Philip Neuhart

Wien, June 2021
Supervisor: Univ.-Prof. Dr. Arnold Neumaier

Statistical Classifiers like the support vector classifier or the k-nearest neighbour classifier have proven to be effective tools for classifying real-world data. In this assignment I would like to develop the main theory behind the Maximal Margin Classifier, the simplest support vector algorithm, and also introduce the k-nearest neighbour algorithm. In the second part I am focusing on the application of these classifiers to a real world problem, handwritten digit recognition. The basis of this experiment was the MNIST data set of handwritten digits, which is publicly available. I used small subsets of 1,5 and 10 examples per digit and created new training examples through shifts and rotations of these images to obtain a larger training set, and thus, hopefully improved test accuracy. Furthermore, I tried to find small training sets, called super-trainers, which give rise to particularly good test performances. Finally, I combined my preprocessing approaches and was able to score test accuracy rates significantly higher than the mean score of randomly assembled training sets.

Contents

1 Introduction

This chapter is based on Cristianini, Shawe-Taylor [3], James et al. [9] and Marques de Sá [13].

In contrary to regression models, statistical classification models try to predict a qualitative/categorical output y given an input x. For example, assuming there are only blue, green and brown eyes, one could, try to predict the eye colour of every student on campus given their hair colour and the country they were born in. In general, this setting can be modelled mathematically by defining an input space (sometimes also referred to as feature space) $X \subseteq \mathbb{R}^n$ and an output domain $Y = \{1, ..., m\}, m \geq 2$. If $m = 2$, we deal with so-called binary classification, otherwise we work in the domain of m-class classification. The elements of X are called observations or instances, whereas the elements of Y are referred to as classes or labels. The dimension of X equals the number of attributes/features of the inputs. In our previous example we have $n = 2$, i.e., hair colour and country. Because we assumed, for simplicity, that there are only three different eye colours, we have $m = 3, Y = \{1, 2, 3\}$. Now, let's assume that we collected the data (eye colour, hair colour and native country) of 100 students already. This set of data S is usually called training data and can be expressed as,

$$S = ((x_1, y_1), ..., (x_l, y_l)) \subseteq (X \times Y)^l,$$

whereas in our example $l = 100$.

Statistical classification models, also known as classifiers, use the information contained in the training data to form a decision rule for classifying new unlabeled input data, also referred to as test data. In our case, although it would almost certainly contradict our training data, a primitive decision rule could be given by saying that each student with brown hair has to have brown eyes as well and every other student has blue eyes. Usually, classifiers take a more sophisticated approach but the idea of calibrating a decision rule by using the training data remains the same.

In subsequent chapters we will take a closer look at a well-known classification model, the Support Vector Machines (SMVs). But before that we need to develop some important tools from optimization theory. We will also introduce a simpler approach, called Linear Learning Machines, to lay the foundation for the more complex SVMs.

2 Support Vector Machines

This chapter is based on [3]. The definitions, propositions and theorem are quoted from there.

2.1 Optimisation Theory

In this section we will provide tools from optimisation theory necessary to be able to analyse and understand concepts like the Maximum Margin Classifier. First we will give some basic definitions to define the mathematical setting relevant for our applications.

Definition 1. [3](p.80) (Primal optimisation problem) Given functions $f, g_i, i = 1, \dots, k$ and $h_i, i = 1, \dots, m$, defined on a domain $\Omega \subseteq \mathbb{R}^n$,

$$\begin{array}{lll} \text{minimise} & f(\mathbf{w}), & \mathbf{w} \in \Omega, \\ \text{subject to} & g_i(\mathbf{w}) \leq 0, & i = 1, \dots, k, \\ & h_i(\mathbf{w}) = 0, & i = 1, \dots, m, \end{array}$$

where $f(\mathbf{w})$ is called the objective function, and the constraints are called, respectively, the inequality and equality constraints. The value of the optimisation problem is defined to be the optimal value of the objective function.

For simplicity we will use $\mathbf{g}(\mathbf{w}) \leq \mathbf{0}$ to denote $g_i(\mathbf{w}) \leq 0$, $\forall i = 1, \dots, k$ (analogously for $\mathbf{h}(\mathbf{w}) = \mathbf{0}$). An inequality constraint $g_i(w) \leq 0$ is described as active (or tight) if the solution (of the optimisation problem) w^* satisfies $g_i(w^*) = 0$, otherwise it is said to be inactive for w^*.

Definition 2. [3](p.80) An optimisation problem for which the objective function is quadratic, while the constraints are all linear, is called quadratic programm.

Definition 3. [3](p.81) An optimisation problem for which the set Ω, the objective function and all of the constraints are convex is said to be convex.

From now on we will restrict ourselves to convex quadratic programmes. Next, we will state some important results from Lagrangian theory that will provide the foundation for developing efficient solutions for the optimisation task in SVMs.

Definition 4. [3](p.83) Given an optimisation problem with objective function $f(\mathbf{w})$, and only equality constraints $h_i(\mathbf{w}) = 0, i = 1, \dots, m$, we define the Lagrangian function as

$$L(\mathbf{w}, \boldsymbol{\beta}) = f(\mathbf{w}) + \sum_{i=1}^{m} \beta_i h_i(\mathbf{w})$$

where the coefficients β_i are called the Lagrange multipliers.

Theorem 2.1. *[3](p.83) (Lagrange) A necessary condition for a point $\mathbf{w}^*$ to be a minimum of $f(\mathbf{w})$ subject to $h_i(\mathbf{w}) = 0, i = 1, \ldots, m$, with $f, h_i \in C^1$, is*

$$\frac{\partial L(\mathbf{w}^*, \boldsymbol{\beta}^*)}{\partial \mathbf{w}} = \mathbf{0}$$

$$\frac{\partial L(\mathbf{w}^*, \boldsymbol{\beta}^*)}{\partial \beta} = 0$$

for some values $\boldsymbol{\beta}^*$. If $L(\mathbf{w}, \boldsymbol{\beta}^*)$ is a convex function of $\mathbf{w}$, the above conditions are also sufficient.

Definition 5. [3](p.85) Given an optimisation problem with domain $\Omega \subseteq \mathbb{R}^n$,

$$\begin{array}{lll} \text{minimise} & f(\mathbf{w}), & \mathbf{w} \in \Omega, \\ \text{subject to} & g_i(\mathbf{w}) \leq 0, & i = 1, \ldots, k, \\ & h_i(\mathbf{w}) = 0, & i = 1, \ldots, m, \end{array}$$

we define the generalised Lagrangian function as

$$\begin{aligned} L(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) &= f(\mathbf{w}) + \sum_{i=1}^{k} \alpha_i g_i(\mathbf{w}) + \sum_{i=1}^{m} \beta_i h_i(\mathbf{w}) \\ &= f(\mathbf{w}) + \alpha' \mathbf{g}(\mathbf{w}) + \boldsymbol{\beta}' \mathbf{h}(\mathbf{w}) \end{aligned}$$

We can now define the Lagrangian dual problem.

Definition 6. [3](p.85) The Lagrangian dual problem of the primal problem of Definition 1 is the following problem:

$$\text{maximise } \theta(\alpha, \beta)$$
$$\text{subject to } \alpha \geq 0$$

where $\theta(\alpha, \beta) = \inf_{w \in \Omega} L(w, \alpha, \beta)$. The value of the objective function at the optimal solution is called the value of the problem.

Dual problems are often easier solved than the primal problem because inequality constraints are in general more difficult to handle directly. We will now state important results regarding the relation of the primal and dual problem. The following theorem shows that the value of the dual problem is always smaller or equal to the primal problem.

Theorem 2.2. *[3](p.85) Let $\mathbf{w} \in \Omega$ be a feasible solution of the primal problem of Definition 1 and (α, β) a feasible solution of the dual problem defined above. Then $f(\mathbf{w}) \geq \theta(\boldsymbol{\alpha}, \boldsymbol{\beta})$.*

Proof. From the definition of $\theta(\alpha, \beta)$ for $\mathbf{w} \in \Omega$ we have

$$\begin{aligned}\theta(\boldsymbol{\alpha}, \boldsymbol{\beta}) &= \inf_{\boldsymbol{u} \in \Omega} L(\mathbf{u}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \\ &\leq L(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \\ &= f(\mathbf{w}) + \alpha' \mathbf{g}(\mathbf{w}) + \boldsymbol{\beta}' \mathbf{h}(\mathbf{w}) \leq f(\mathbf{w})\end{aligned}$$

since the feasibility of $\mathbf{w}$ implies $\mathbf{g}(\mathbf{w}) \leq \mathbf{0}$ and $\mathbf{h}(\mathbf{w}) = \mathbf{0}$, while the feasibility of (α, β) implies $\alpha \geq 0$. □

Corollary 2.3. *[3](p.85) If $f(\mathbf{w}^*) = \theta(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$, where $\boldsymbol{\alpha}^* \geq \mathbf{0}$, and $\mathrm{g}(\mathbf{w}^*) \leq \mathbf{0}, \mathbf{h}(\mathbf{w}^*) = \mathbf{0}$, then $\mathbf{w}^*$ and $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ solve the primal and dual problems respectively. In this case $\alpha_i^* g_i(\mathbf{w}^*) = 0$, for $i = 1, \ldots, k$.*

Proof. If $f(w^*) = \theta(\alpha^*, \beta^*)$, then the inequalities in the proof of theorem 2.2 become equalities. Hence, $\alpha^* g(w^*) = 0$ □

In general, the value of the primal and the dual problem need not be the same. The difference between the values of the primal and dual problems is known as the **duality gap**. The strong duality theorem provides conditions for when the duality gap is zero. This is an important result, since it allows us to find the value of the primal problem by solving the dual problem, which in the case of optimising SVMs is easier solved than its corresponding primal problem (see subsection 2.4).

Theorem 2.4. *[3](p.86) (Strong duality theorem) Given an optimisation problem with convex domain $\Omega \subseteq \mathbb{R}^n$,*

$$\begin{array}{lll} \textit{minimise} & f(\mathbf{w}), & \mathbf{w} \in \Omega, \\ \textit{subject to} & g_i(\mathbf{w}) \leq 0, & i = 1, \ldots, k, \\ & h_i(\mathbf{w}) = 0, & i = 1, \ldots, m, \end{array}$$

where the g_i and h_i are affine functions, that is

$$\mathbf{h}(\mathbf{w}) = \mathbf{A}\mathbf{w} - \mathbf{b}$$

for some matrix $\mathbf{A}$ and vector $\mathbf{b}$, the duality gap is zero.

We are now ready to state the main theorem of this section, called the Kuhn-Tucker Theorem.

Theorem 2.5. *[3](p.87) (Kuhn-Tucker) Given an optimisation problem with* *convex domain* $\Omega \subseteq \mathbb{R}^n$

$$\begin{array}{lll} \textit{minimise} & f(\mathbf{w}), & \mathbf{w} \in \Omega, \\ \textit{subject to} & g_i(\mathbf{w}) \leq 0, & i = 1, \ldots, k, \\ & h_i(\mathbf{w}) = 0, & i = 1, \ldots, m, \end{array}$$

with $f \in C^1$ *convex and* g_i, h_i *affine, necessary and sufficient conditions for a point* w^* *to be an optimum are the existence of* α^*, β^* *such that*

$$\begin{aligned} \frac{\partial L(\mathbf{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \mathbf{w}} &= \mathbf{0}, \\ \frac{\partial L(\mathbf{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \beta} &= 0, \\ \alpha_i^* g_i(\mathbf{w}^*) &= 0, i = 1, \ldots, k \\ g_i(\mathbf{w}^*) &\leq 0, i = 1, \ldots, k, \\ \alpha_i^* &\geq 0, i = 1, \ldots, k. \end{aligned}$$

The KKT conditions say that either a constraint is active, meaning $g_i(w^*) = 0$, or the corresponding multiplier satisfies $\alpha_i^* = 0$. This is summarised in the equation $g_i(w^*)\alpha_i^* = 0$. Hence, only active constraints have non-zero dual variables. This observation will later give rise to the term 'support vector', which is used for training examples for which the dual variables are non-zero.

2.2 Linear Learning Machines

linear classifiers pose the foundation of more complex tools like support vector machines. One major advantage of linear classifiers is that they are computationally efficient when compared to most non-linear approaches and have even shown to be able to compete with them in performance in some applications [17].

Definition 7. For convenience, we define the sign function as follows

$$\operatorname{sign}(x) := \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

Linear learning machines work for binary classification (there are mulit-class variations) of data by calibrating an affine linear function $f : X \subseteq \mathbb{R}^n \to \mathbb{R}$ on the input space X such that the decision rule $h(x) := \operatorname{sign}(f(x))$ separates the data according to their label,

$$
\begin{aligned}
f(x) &= \langle w, x \rangle + b \\
&= \sum_{i=1}^{n} w^i x^i + b,
\end{aligned}
$$

where $(w, b) \in \mathbb{R}^n \times \mathbb{R}$ are parameters that control the function. We will from now on associate every function with its parameters (w, b). Each $x \in X$ is either assigned to the positive class, if $\text{sign}(f(x)) \geq 0$, or to the negative class otherwise.

Geometrically, the affine linear function f corresponds to an affine hyperplane defined by the equation $\langle w, x \rangle + b = 0$ (see Figure 1). A geometric interpretation

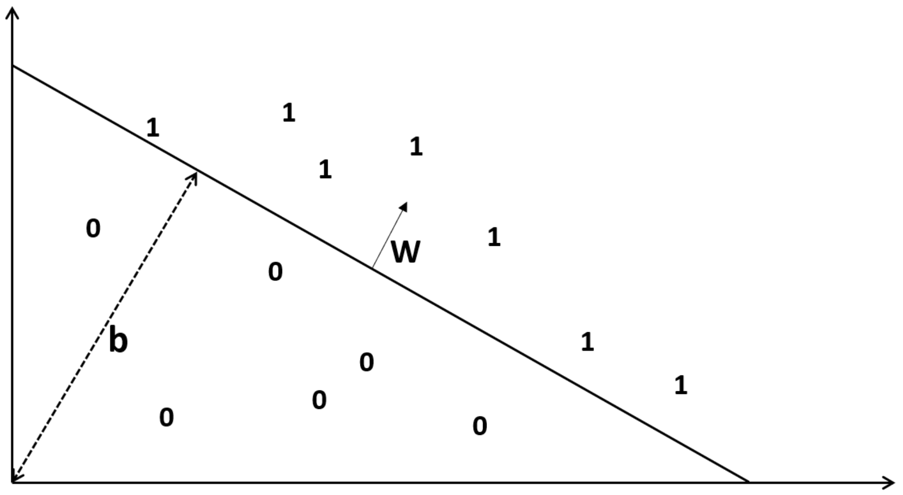


Figure 1: A seperating hyperplane (w, b) for $dim(X) = 2$

is that the input space X is split into two parts by the affine hyperplane defined by the equation $\langle w, x \rangle + b = 0$ (see Figure 1). For further reading on the geometric interpretation of the hyperplane and its parameters (w, b) see [14].

The decision function can also be rewritten in dual coordinates [3]:

$$
\begin{aligned}
h(x) &= \text{sign}(\langle w, x \rangle + b) \\
&= \text{sign}\left(\left\langle \sum_{j=1}^{l} \alpha_i y_i x_i, x \right\rangle + b \right) \\
&= \text{sign}\left(\sum_{j=1}^{l} \alpha_i y_i \langle x_i, x \rangle + b \right),
\end{aligned}
$$

with $w = \sum_{j=1}^{l} \alpha_i y_i x_i$. Note that the decision rule is using only inner products between training examples and the test observation.

2.3 Kernel-Induced Feature Spaces

2.3.1 Learning in Feature Space

To be able to work with linear classifiers we require the training data to be linearly separable, i.e. the classes can be separated by a hyperplane in the input space. But in real-world applications this is often not the case. To be able to circumvent this problem, a common preprocessing strategy is to map the input space X into a new space, $F = \{\phi(x) : x \in X\}$:

$$x = (x_1, ..., x_n) \mapsto \phi(x) = (\phi_1(x), ..., \phi_N(x)).$$

The components of x are usually called attributes, whereas the components of $\phi(x)$ are called features. The space X is referred to as the input space, while F is called the feature space. Wilimitis gives a simple but illustrative example of such a mapping [19], see Figure 2. It shows a situation where the data is linearly inseparable in the input space but when projected onto a two-dimensional feature space by the map $\phi(x) = x^2$, we obtain linearly separable data.

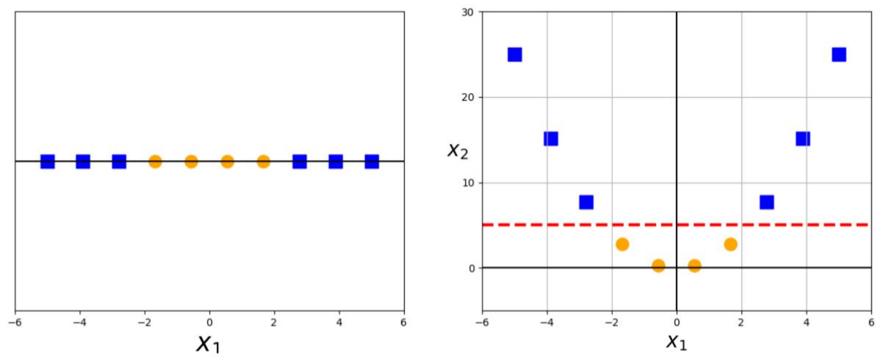


Figure 2: [19] A feature map can simplify the classification task

2.3.2 The Implicit Mapping into Feature Space

In order to classify 'real-world' data which is often not linearly separable we need to find a more sophisticated approach than to just try to separate the data by a hyperplane in the input space. But at the same time we do now want to give up the computational efficiency of linear classifiers. A brilliant idea, suggested by Vapnik et al. in 1992 [2], is to use the so-called kernel trick. First, we extend our hypothesis space to

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i \phi_i(\mathbf{x}) + b$$

where $\phi : X \to F$ can be a non-linear map from the input space to some feature space. If we manage to find a suitable non-linear map ϕ, we obtain linearly separable data in the feature space which can thus be classified using a linear classifier in the feature space.

As mentioned before in 2.2, one important property of linear machines is that they can be expressed in a dual representation:

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} \alpha_i y_i \left\langle \phi\left(\mathbf{x}_i\right) \cdot \phi(\mathbf{x}) \right\rangle + b$$

If we can compute the inner product $\langle \phi\left(\mathbf{x}_i\right) \cdot \phi(\mathbf{x}) \rangle$ directly as a function of the original inputs x_i, we can classify the data in the feature space, without having to bear the computational costs of evaluating ϕ on each training example.

Definition 8. [3](p.30) A kernel is a function $K : X \times X \to \mathbb{R}$, such that for all $\mathbf{x}, \mathbf{z} \in X$

$$K(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle$$

where ϕ is a mapping from X to an (inner product) feature space F.

The only information used about the training examples is stored in their Gram matrix $\mathbf{K}$, which in this context is also referred to as the kernel matrix. Its entries are defined by $\mathbf{K_{ij}} := K(x_i, x_j)$. We will denote it with the symbol $\mathbf{K}$. The goal is to find a kernel function that can be evaluated efficiently. Once we found it, the decision rule can be evaluated by at most ℓ evaluations of the kernel:

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} \alpha_i y_i K\left(\mathbf{x}_i, \mathbf{x}\right) + b$$

Not that, when using a kernel, we do not even need to know the underlying feature map explicitly to be able to perform a classification in the feature space. To illustrate this further we take a look at the following example:

Example. Consider the following kernel

$$\begin{aligned}
K(x,z) = (\langle x \cdot z\rangle + c)^2 &= \left(\sum_{i=1}^{n} x_i z_i + c\right)\left(\sum_{j=1}^{n} x_j z_j + c\right) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j z_i z_j + 2c\sum_{i=1}^{n} x_i z_i + c^2 \\
&= \sum_{(i,j)=(1,1)}^{(n,n)} (x_i x_j)(z_i z_j) + \sum_{i=1}^{n}\left(\sqrt{2c}x_i\right)\left(\sqrt{2c}z_i\right) + c^2
\end{aligned}$$

whose $\binom{n+1}{2} + n + 1 = \binom{n+2}{2}$ features are all the monomials of degree up to 2. The parameter c controls the relative weightings between the degrees 1 and 2, and the strength of the constant feature.

Note, that we did not have to state the feature map ϕ given by,

$$\phi(x) = \begin{pmatrix} x_1 x_1 \\ x_1 x_2 \\ \vdots \\ x_n x_n \\ \sqrt{2}c x_1 \\ \vdots \\ \sqrt{2}c x_n \\ c \end{pmatrix}$$

explicitly to be able to define the kernel function. Furthermore, evaluating the kernel requires only $O(n)$ operations, whereas first evaluating ϕ on x and z and then computing the inner product requires $O(n^2)$ operations.

2.3.3 The Characterisation of Kernels

Mercer's theorem provides a characterisation of when a function $K(\mathbf{x}, \mathbf{z})$ is a kernel.

Proposition 2.6. *[3](p.33) (Mercer) Let X be a finite input space with $|X| = n$. Then $K(\mathbf{x}, \mathbf{z})$ is a kernel function if and only if the matrix*

$$\mathbf{K} = \left(K\left(\mathbf{x}_i, \mathbf{x}_j\right)\right)_{i,j=1}^{n}.$$

is symmetric and positive semi-definite (has only non-negative eigenvalues).

2.3.4 Examples for Kernels

There are a wide variety of kernels used but the most common ones are the linear kernel, rbf kernel and polynomial kernel which we are going to use later on in chapter 3 for training the support vector classifier on the MNIST data set of handwritten digits.

Definition 9. Let $X \subseteq \mathbb{R}^n$ be the input space. A kernel function $K : X \times X \to \mathbb{R}$ is called a

$$\begin{aligned} \text{Linear kernel} &\iff K(x,z) = x^T z + c = \langle x, z \rangle + c \\ \text{Radial basis function (rbf) kernel} &\iff K(x,z) = exp(-\gamma \|x - z\|^2) \\ \text{Polynomial kernel} &\iff K(x,z) = (x^T z + c)^d, \end{aligned}$$

where $\gamma > 0$, $c \in \mathbb{R}$ and $d \in \mathbb{Z}^+$ are parameters.

The linear kernel, also referred to as non-kernel, is the simplest kernel of all [7]. Note that it does not project the data onto a higher dimensional feature space. Therefore it can only be used for linearly separable input data.

2.4 Support Vector Classification

The support vector classification aims to devise a computationally efficient method of finding a hyperplane in a high dimensional feature set, which seperates the training data. We will use the results of subsection 2.3 and 2.1 to derive the main theory of support vector classification, although there is no attempt to be exhaustive.

2.4.1 The Maximal Margin Classifier

The simplest version of SVMs is the so-called Maximal Margin Classifier. The downside of this model is that it only works for linearly separable data in the feature space. Nevertheless it is a very useful introduction and it forms the main building block for more complex approaches.
The strategy of the maximal margin classifier is to find the maximal margin hyperplane in an appropriately chosen kernel-induced feature space. It is implemented by reducing the problem to a convex optimisation problem.

Let (w, b) be a separating hyperplane for a training set $S \subseteq (X \times Y)^l$. Recall that the functional margin γ_i^f of a training example $(x_i, y_i) \in S$ is defined as

$$\gamma_i^f = y(\langle w \cdot x \rangle + b),$$

and its geometric margin $\gamma_i = \gamma_i^g$ is given by

$$\gamma_i = y(\langle \frac{w}{\|w\|} \cdot x \rangle + \frac{b}{\|w\|}) = \frac{1}{\|w\|} \gamma_i^f.$$

The geometric margin measures the Euclidean distance of the training examples to the hyperplane. Furthermore, the functional margin γ^f of the hyperplane (w, b) is the minimum of the functional margin distribution. Similarly, the (geometric) margin γ of the hyperplane is given by the minimum of the geometric margin distribution. We observe that the following relation holds

$$\gamma = \frac{1}{\|w\|} \gamma^f.$$

Note that by rescaling the hyperplane to $(\lambda w, \lambda b)$, for $\lambda \in \mathbb{R}^+$, we do not change the hyperplane itself. Thus, the geometric does not change either. However, the functional margin gets scaled by λ. We take advantage of this inherent degree of freedom by fixing the functional margin γ^f to be equal to 1. Hence, the geometric margin is $\gamma = \frac{1}{\|w\|}$.

Finally, the maximal margin hyperplane is the hyperplane realising the maximum geometric margin over all hyperplanes, i.e. it has the greatest minimum distance to the training points.

Proposition 2.7. *[3](p.95) Given a linearly separable training sample*

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell))$$

the hyperplane $(\mathbf{w}, b)$ *that solves the optimisation problem*

$$\begin{array}{ll} \textit{minimise}_{\ \mathbf{w},b} & \langle \mathbf{w} \cdot \mathbf{w} \rangle \\ \textit{subject to} & y_i (\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1 \\ & i = 1, \ldots, \ell \end{array}$$

realises the maximal margin hyperplane with geometric margin $\gamma = 1/\|\mathbf{w}\|_2$.

Proof. As already stated above, we can make use of the fact that rescaling a hyperplane (w, b) to $(\lambda w, \lambda b)$ does not change its geometric margin but scales its functional margin by the same parameter λ. We can therefore choose our maximal margin hyperplane $(\tilde{w}, \tilde{b})$ to have a functional margin $\tilde{\gamma}^f$ of 1 on our training set S. Hence, the geometric margin $\tilde{\gamma}$ of $(\tilde{w}, \tilde{b})$ is equal to $\frac{1}{\|\tilde{w}\|}$. Now, assume that for the solution (w^*, b^*) of the above optimisation the constraints are inactive:

$$y_i (\langle w \cdot x_i \rangle + b) = \lambda_i > 1.$$

We define $\lambda := \min\limits_{1\le i\le l} \lambda_i$ and observe that $\forall\ 1 \le i \le l$,

$$y_i\left(\left\langle \frac{w}{\lambda}\cdot x_i\right\rangle + \frac{b}{\lambda}\right) \ge 1,$$

and $\exists j \in \{1, ..., l\}$ such that the constraints are active,

$$y_j\left(\left\langle \frac{w}{\lambda}\cdot x_j\right\rangle + \frac{b}{\lambda}\right) = 1.$$

But since $\langle \frac{w}{\lambda}\cdot\frac{w}{\lambda}\rangle = \frac{1}{\lambda^2}\langle w\cdot w\rangle < \langle w\cdot w\rangle$ we obtain a contradiction. Therefore, at least one constraint must be active. Hence, the functional margin γ^f is equal to 1. We observe that

$$\frac{1}{\|w^*\|} = \gamma \le \tilde{\gamma} = \frac{1}{\|\tilde{w}\|}.$$

But since $(\tilde{w}, \tilde{b})$ is also a feasible solution to the above optimisation problem, it follows that

$$\|w^*\|^2 = \langle w^*\cdot w^*\rangle \le \langle \tilde{w}\cdot\tilde{w}\rangle = \|\tilde{w}\|^2,$$

or equivalently,

$$\frac{1}{\|w^*\|} \ge \frac{1}{\|\tilde{w}\|}.$$

Finally, we obtain that $\frac{1}{\|w^*\|} = \frac{1}{\|\tilde{w}\|}$ and $\gamma = \tilde{\gamma}$. In other words, (w^*, b^*) realises the maximum margin hyperplane with geometric margin $\frac{1}{\|w^*\|}$. $\square$

In the next step we want to transform this optimisation problem into its corresponding dual problem. The primal Lagrangian is

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\langle\mathbf{w}\cdot\mathbf{w}\rangle - \sum_{i=1}^{\ell}\alpha_i\left[y_i\left(\langle\mathbf{w}\cdot\mathbf{x}_i\rangle + b\right) - 1\right]$$

where $\alpha_i \ge 0$ are the Lagrange multipliers, as described in subsection 2.1 . The corresponding dual is found by differentiating with respect to $\mathbf{w}$ and b,

$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial\mathbf{w}} = \mathbf{w} - \sum_{i=1}^{\ell} y_i\alpha_i\mathbf{x}_i = \mathbf{0}$$

$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial b} = \sum_{i=1}^{\ell} y_i\alpha_i = 0$$

and resubstituting the relations obtained,

$$\mathbf{w} = \sum_{i=1}^{\ell} y_i\alpha_i\mathbf{x}_i,$$

$$0 = \sum_{i=1}^{\ell} y_i \alpha_i$$

into the primal to obtain

$$\begin{aligned}
W(\alpha) := L(\mathbf{w}, b, \boldsymbol{\alpha}) &= \frac{1}{2}\langle \mathbf{w} \cdot \mathbf{w} \rangle - \sum_{i=1}^{\ell} \alpha_i \left[y_i \left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \right) - 1 \right] \\
&= \frac{1}{2} \sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle - \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle + \sum_{i=1}^{l} \alpha_i \\
&= \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle
\end{aligned}$$

Remark. The application of the optimisation tools developed in 2.1 naturally leads to the dual representation. The dual representation is required for the use of kernels since we need the hypothesis to be described by a linear combination of training instances.

From Proposition 2.7 and Theorem 2.5 we obtain the following proposition:

Proposition 2.8. *[3](p.96) Consider a linearly separable training sample*

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell))$$

and suppose the parameters α^* solve the following quadratic optimisation problem:

$$\begin{array}{ll}
\textit{maximise} & W(\alpha) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\prime} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle \\
\textit{subject to} & \sum_{i=1}^{l} y_i \alpha_i = 0 \\
 & \alpha_i \geq 0, i = 1, \ldots, \ell
\end{array}$$

Then the weight vector $\mathbf{w}^* = \sum_{i=1}^{\ell} y_i \alpha_i^* \mathbf{x}_i$ realises the maximal margin hyperplane with geometric margin

$$\gamma = 1 / \|\mathbf{w}^*\|_2$$

Remark. The value of b does not appear in the dual problem and so b^* must be found making use of the primal constraints:

$$b^* = -\frac{\max_{y_i=-1} \left(\langle \mathbf{w}^* \cdot \mathbf{x}_i \rangle \right) + \min_{y=-1} \left(\langle \mathbf{w}^* \cdot \mathbf{x}_i \rangle \right)}{2}$$

The KKT complementarity conditions from this optimisation problem state that the optimal solutions α^*, $(\mathbf{w}^*, b^*)$ must satisfy

$$\alpha_i^* \left[y_i \left(\langle \mathbf{w}^* \cdot \mathbf{x}_i \rangle + b^* \right) - 1 \right] = 0, \quad i = 1, \ldots, \ell.$$

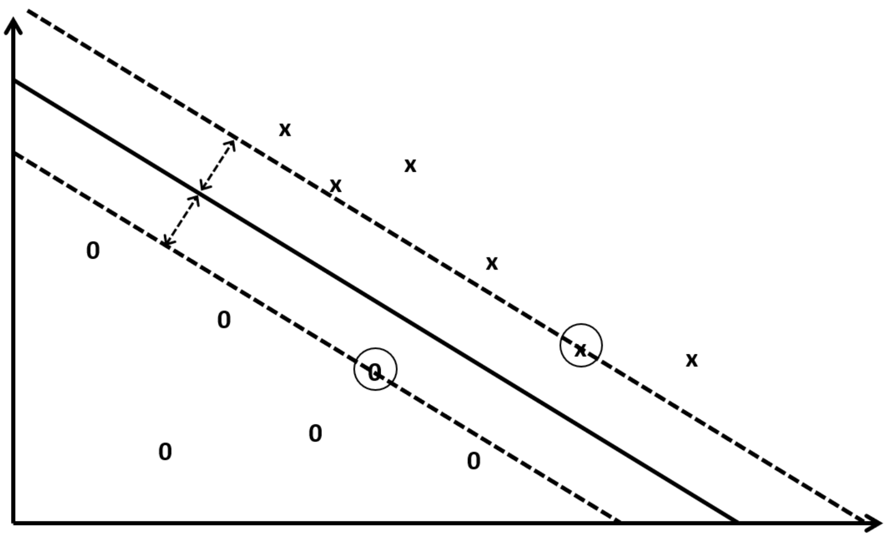


Figure 3: A maximal margin hyperplane with its support vectors

This implies that only for inputs x_i for which the functional margin is one and that therefore lie closest to the hyperplane are the corresponding α^* non-zero. All the other parameters α^* are zero. This means that only the training examples x_i for which the corresponding Lagrange multipliers α_i are non-zero have an influence on the weight vector $w*$, and thus on the separating hyperplane, see Figure 3. These training examples are called support vectors. We will denote the set of indices of the support vectors with sv. Furthermore the optimal hyperplane can be expressed in the dual representation in terms of this subset of the parameters:

$$\begin{aligned} f(x, \alpha^*, b^*) &= \sum_{i=1}^{l} y_i \alpha_i^* \langle x_i \cdot x \rangle + b^* \\ &= \sum_{i \in sv} y_i \alpha_i^* \langle x_i \cdot x \rangle + b^* \end{aligned}$$

Training examples that are not support vectors have no influence. Therefore, removing them will not affect the solution.

Finally, we can state the main proposition of this chapter. We can make use of kernels, results from optimisation theory and the theory we developed about linear classifiers. This is possible thanks to a remarkable property of both the dual objective from Proposition 2.8 and the decision function defined in subsection 2.2, that the training data only appears in the inner product.

Proposition 2.9. *[3](p.98) Consider a training sample*

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell))$$

that is linearly separable in the feature space implicitly defined by the kernel $K(\mathbf{x}, \mathbf{z})$ *and suppose the parameters* α^* *and* b^* *solve the following quadratic optimisation problem:*

$$\begin{aligned} &\textit{maximise } W(\alpha) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{ij=1}^{\prime} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j), \\ &\textit{subject to } \sum_{i=1}^{\ell} y_i \alpha_i = 0 \\ &\qquad\qquad\ \alpha_i \geq 0, i = 1, \ldots, \ell \end{aligned}$$

Then the decision rule given by $\operatorname{sign}(f(\mathbf{x}))$*, where* $f(\mathbf{x}) = \sum_{i \in sv} y_i \alpha_i^* K(\mathbf{x}_i, \mathbf{x}) + b^*$*, is equivalent to the maximal margin hyperplane in the feature space implicitly defined by the kernel* $K(\mathbf{x}, \mathbf{z})$.

Remark. Although there are ways to force separability for any training set in an accordingly chosen feature space it is most often not desirable to do so, since this approach leads to overfitting. Instead, one could allow some misclassifications in the learning process, i.e., not force the training data to be linearly separable in the feature space. Although out of the scope of this assignment, the soft margin classifier poses a well-known example for such an approach [3].

Remark. A common way of generalizing the binary classification theory to the multi-class case is to use the One-vs-all approach. Given an m-class training set S. For each class $j \in \{1, ..., m\}$, the classifier is trained on a binary training set, formed by aggregating the examples from all classes, except class j, to one single class. This way, one obtains m hypothesis. In the case of linear classifiers, and thus also for SVMs, for a given input, the decision rule is then given by assigning the class, for which the corresponding hypothesis outputs the maximum value.

3 Application

We are now going to see an application of support vector machines. Nielsen poses in his article 'Reduced MNIST: how well can machines learn from small data?' [15] the interesting question of how well we can expect machines to learn from relatively small data sets. Our goal will be to use small subsets of the MNIST database to analyse how well the Support Vector Machines approach performs it. The training sets will contain either 1,5 or 10 examples of each digit from 0 to 9. These training sets are referred to as MNIST/1, MNIST/5 and MNIST/10 respectively, to be consistent with Nielsen's notation. To increase the test accuracy we are going to artificially extend our training sets by rotating and shifting the images to generate new observations. Furthermore, we will compare the results to a simpler but nevertheless competitive classifier, called the k-nearest neighbour algorithm. All classifier algorithms I used for this project are taken from the `sklearn` machine learning library [18].

3.1 k-Nearest Neighbors

This section is based on [9].

We are going to compare our results for the Support Vector Machines with a simpler approach, called the k-nearest neighbor algorithm. It was first published by Fix and Hodges in 1951 [8].
Given a training set $S = \{(x_i, y_i)\}_{i=1}^{l} \subseteq (X \times Y)^l$, the kNN-approach is to classify an previously unknown (unlabeled) instance x by estimating the probabilities of x belonging to each class $j \in Y$, and assigning it to the class corresponding to the highest probability. The kNN algorithm first identifies the set N_x^k of k nearest training examples of x, where $k \in \mathbb{Z}^+$ is a parameter, and then estimating the conditional probability for class j by computing the fraction of training examples in N_x^k who also belong to this class,

$$P(Y = j | X = x) = \frac{1}{k} \sum_{x_i \in N_x^k} \delta_{y_i j}.$$

It is important to notice that N_x^k depends on the metric used to measure the distances between observations in X. The most common choice is the standard Euclidean distance but in general any metric can be used [4].

In figure 4 we can see a simple but illustrative example of the application of the k-nearest neighbor algorithm. On the left-hand panel we can see the two-dimensional ball containing the three nearest neighbours of x, measured in the standard Euclidean metric. The conditional probabilities for the blue and the yellow class are $\frac{2}{3}$ and $\frac{1}{3}$ respectively since the ball contains two blue training examples but only a single yellow example. Hence, the algorithm will assign x to the blue class. The right-hand panel shows the decision boundary in black. Test observations will be assigned to the blue class if they lie in the blue coloured area and to the yellow class otherwise.

3.2 MNIST Dataset

This section is based on [6] and [12].

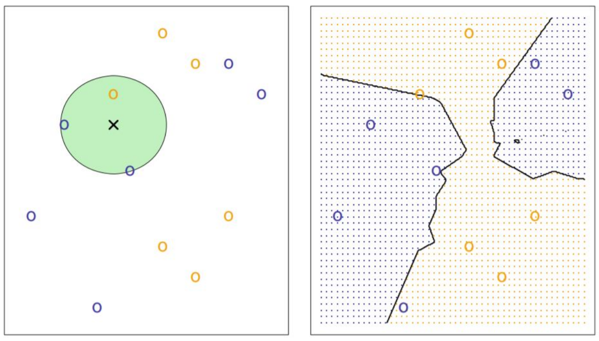


Figure 4: [9] (p.40). The kNN approach illustrated with two classes, K=3 and the Euclidean distance.

The database of 42,000 labeled training examples of handwritten digits used for this exercise is based on the MNIST database [12] and is available at [6], which comprises a training set of 60,000 examples and a test set of 10,000 examples. The MNIST database is optimally suited for beginners as the data does not need a lot of preprocessing before one can achieve reasonable accuracy results. The digits have been size-normalized and centered in a fixed-size image. Each example consists of 784 (28x28) pixels representing a black and white image of a handwritten digit, from zero through nine. Each pixel has a value associated with it, representing the gray scale of the pixel and ranging from 0 to 255. The .csv-file containing the data is structured as a table, where the first column represents the labels of each example and columns 2-785 contain the pixel values for each example (see Table 1).

Label	Pixel0	Pixel1	...	Pixel783
1	0	30	...	0
4	60	110	...	0
9	211	240	...	32
⋮	⋮	⋮	⋮	⋮

Table 1: Structure of the training set

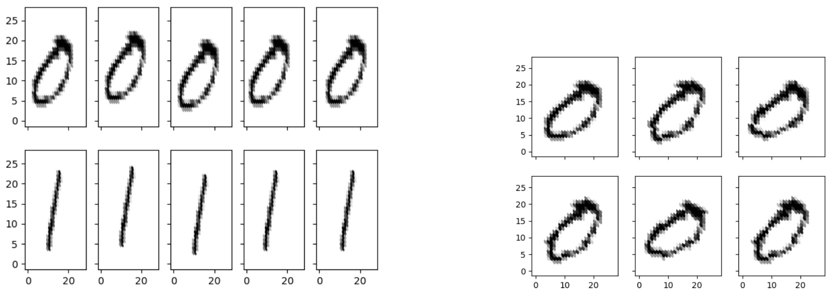
Table 2 shows benchmarks for different classifiers given that the classifier model was trained on the whole MNIST database training set of 60,000 examples. Both the SVMs and kNN approach score almost competitive results given appropriate preprocessing of the data, when compared to state of the art convolutional neural network (CNN) approaches.

Classifier	Preprocessing	TER(%)
K-nearest-neighbors, Eucl. (L2)	none	3.09
K-NN, shape context matching	shape context feature extraction	0.63
SVM, Gaussian Kernel	none	1.4
Virtual SVM, deg-9 poly, [...]	deskewing	0.56
committee of 35 CNN [...]	width normalization	0.23

Table 2: Benchmarks for test error rates (TER) different classifiers [12].

3.2.1 Extending the Training Data

As mentioned before, we are going to artificially our training data by shifting and rotating the images. The images are shifted along both axis in both directions by 1 and 2 pixels (see left-hand panel in Figure 5). Hence, out of every image in the baseline training set, eight new images are being created. Furthermore, every image is also rotated by an angle of -15 to 15 degrees, to create another 30 additional images per image (see right-hand panel in Figure 5). The code for shifting and rotating the images is provided in the Appendix in Listing 1 and Listing 2. The library used for rotating the images is Pillow (PIL), which is designed for image processing with few pixel formats [16].



(a) Shifts (b) Rotations

Figure 5: Examples of shifts and rotations

3.2.2 Super-trainers

Another interesting question that was posed by Michael Nielsen [15] is: Can we find super-trainers, i.e., small training sets which give rise to particularly good performance? From now on, I will use the term super-trainers for describing either the best available MNIST/1, MNIST/5 or MNIST/10 data set, i.e. the one which scores the best accuracy when used as the training set. To answer the question posed by Nielsen, I defined a subset of 10000 images (1000 examples of each digit) as the pool of possible super-trainers. Each time one example of each digit was chosen randomly out of that pool to build an MNIST/1 data set, which was then tested on the training data which was not contained in it. I sampled 1000 MNIST/1 data sets to find possible super-trainers and to gain insight in the distribution of their performance as training sets. I used different classifiers for that, see table 3. The support vector classifier (SVC) with linear

	SVC - rbf	SVC - linear	SVC - poly	kNN
Max	0.54	0.57	0.44	0.55
Mean	0.42	0.42	0.3	0.42
Var	0.002	0.002	0.002	0.002

Table 3: MNIST/1 Statistic

kernel performed the best, while the polynomial kernel with degree 2 performed the worst. In figure 6 you can see a plot of the best performing data set, the super-trainers, of the Support Vector classifier with linear kernel. It is interesting to notice but not surprising that the depicted digits also resemble very good examples from a human's perspective. The classifier scored an accuracy of 0.57,

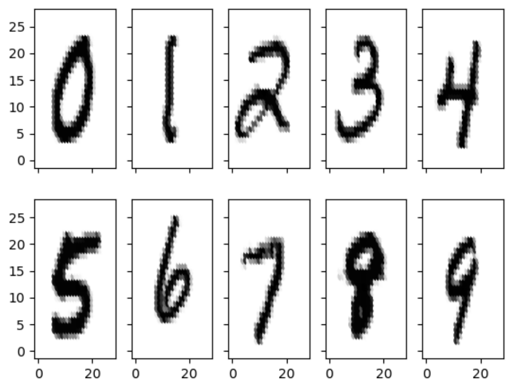


Figure 6: Supertrainers

which means that it classified 57% of all test observations correctly. To get a better understanding of the performance distribution of the 1000 data sets it is useful to look at the corresponding boxplot (see Figure 7). The upper whisker extends to the last data point less than $Q_3 + 1.5 \cdot IQR$, with Q_3 being the upper quartile and IQR being the Interquartile range (Q3-Q1). Note, that the best performing data set is almost 4% better than the second best data set and 15% better than the mean. See the Appendix for the Boxplots of the other kernels and classifiers.

We could aggregate the five best performing MNIST/1 data sets of our sample

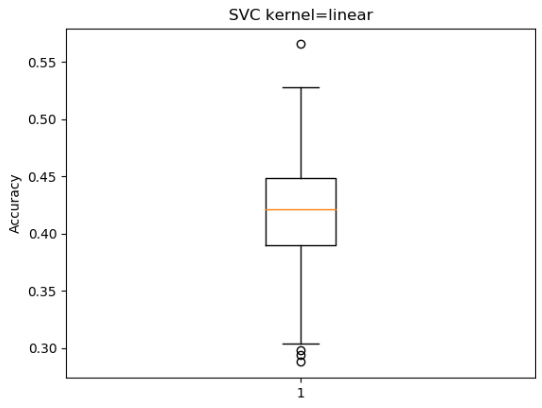


Figure 7: Boxplot of SVC-linear Accuracy

to form a data set of MNIST/5 super-trainers. Analogously, we could form a MNIST/10 set with the top ten MNIST/1 sets. But are these really the best performing MNIST/5 and MNIST/10 data sets we can find? The answer is no. I sampled 500 randomly composed MNIST/5 and MNIST/10 data sets each to see if we can score better, see Figure 8. From the example of the SVC with linear

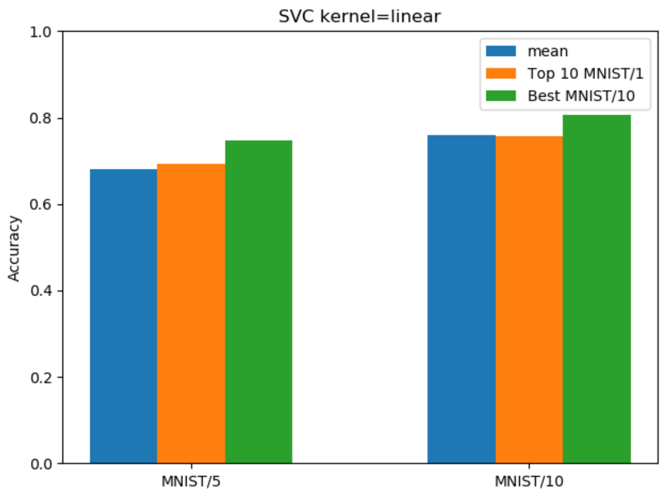
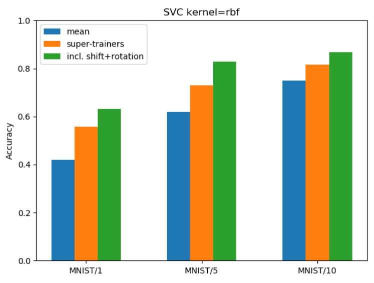


Figure 8: Comparison Top10 MNIST/1 vs. Best MNIST/10

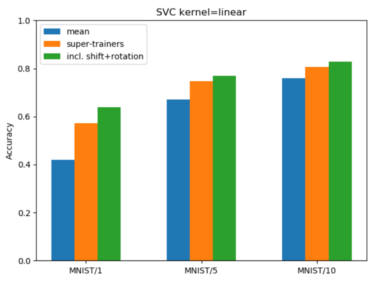
kernel we can see that the Top10 MNIST/1 data set is not even guaranteed to perform above average. This finding gives rise to the conjecture that because the top performing MNIST/1 data sets have very similar structure, aggregating them to a larger training set leads to overfitting. Here, overfitting is used to describe the phenomenon of experiencing a performance drop of our models due to overuse of the same or very similar training data and therefore following the noise contained in the training data too closely. See [5] and [10] for further reading on overfitting.

3.2.3 Final Results

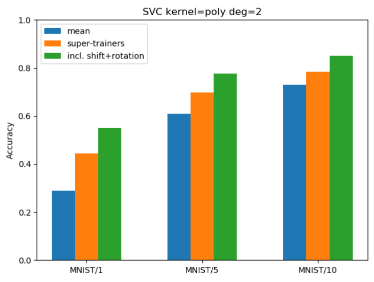
Our goal is now to use our techniques of artificially extending our training sets discussed in subsubsection 3.2.1 to improve our accuracy even further. Figure 9 shows the mean (blue bars) accuracy of randomly composed MNIST/1, MNIST/5 and MNIST/10 data sets and the accuracy boost accomplished by training the classifiers on our super-trainers (orange bars) and finally using small shifts and rotations to create new training observations out of the super-trainers (green bars). The exact figures are represented in table 4. Overall the performance



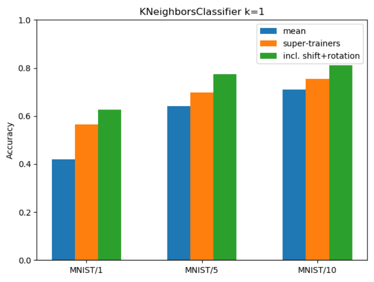
(a) SVC, kernel=rbf



(b) SVC, kernel=linear



(c) SVC, ker=poly, deg=2



(d) kNN, k=1

Figure 9: Performance Increases

has been improved significantly, with accuracy scores up to 87%. It seems plausible to expect that the accuracy of a MNIST/10 data set can be increased to over 90 percent using additional preprocessing tools such as feature extraction, re-scaling etc. See [11] & [1] for further reading on data preprocessing. Although, the performance of the models has improved notably, the results are still far from competitive compared to state of the art machine learning models like neural networks which are trained on far bigger training sets. But considering

	SVC - rbf	SVC - linear	SVC - poly	kNN
MNIST/1				
Mean	0.42	0.42	0.29	0.42
Super-trainers	0.56	**0.57**	0.44	0.55
incl. shifts & rot.	0.63	**0.64**	0.55	0.63
MNIST/5				
Mean	0.64	0.67	0.61	0.62
Super-trainers	0.73	**0.75**	0.70	0.70
incl. shifts & rot.	**0.83**	0.77	0.78	0.77
MNIST/10				
Mean	0.76	0.76	0.73	0.71
Super-trainers	**0.82**	0.81	0.78	0.75
incl. shifts & rot.	**0.87**	0.83	0.85	0.81

Table 4: Performance Statistic

applications where there is simply not more training data available, the results suggest that machine learning models such as SVMs could be able to come reasonably close to their maximum potential using only small data sets.

4 Appendix

4.1 Code

```python
import numpy as np

def img_shift(img_arr, int_shift, axis): #img_arr = input image,
    int_shift = desired pixel shift
    if int_shift==0: return img_arr
    newimage = np.roll(np.reshape(img_arr,(28,28)), int_shift, axis
    =axis)
    if axis == 0:
        if int_shift == 2: newimage[1, :] = np.zeros(28)
        if int_shift >= 1: newimage[0, :] = np.zeros(28)
        if int_shift <= -1: newimage[27, :] = np.zeros(28)
        if int_shift == -2: newimage[26, :] = np.zeros(28)
    if axis == 1:
        if int_shift == 2: newimage[:, 1] = np.zeros(28)
        if int_shift >= 1: newimage[:, 0] = np.zeros(28)
        if int_shift <= -1: newimage[:, 27] = np.zeros(28)
        if int_shift == -2: newimage[:, 26] = np.zeros(28)
    newimage=np.reshape(newimage,784)
    return newimage
```

Listing 1: Code Image Shifting

```python
import numpy as np
from PIL import Image

def img_rotation(arr_img,deg): #arr_img= input image to be rotated,
     rotation= degree of rotation
    pre_img=(np.array(arr_img).reshape(28, 28)).astype(np.uint8) #
    reshaping, etc
    img = Image.fromarray(pre_img, mode='L') #converting to image
    newimage=np.array(img.rotate(deg)).reshape(784) #rotating image
     and converting back to array
    return newimage
```

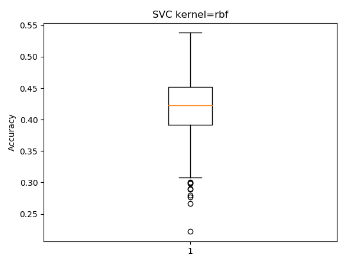
Listing 2: Code Image Rotation

The code used for plotting the images is provided below in Listing 3.

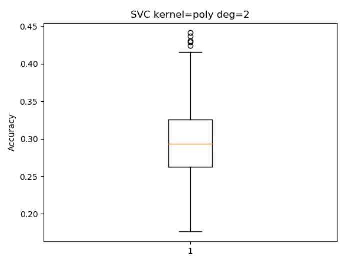
```python
import matplotlib.pyplot as plt
import time

def plotMNIST(Images_l,plot_rows):
    #Images must be a list consisting of 28x28 arrays, plot_rows
    must be a divider of len(Images)
    fig, axs = plt.subplots(plot_rows, int(len(Images_l)/plot_rows)
    )
    for m in range(plot_rows):
        for i in range(int(len(Images_l)/plot_rows)):
            h_map_x=[]
            h_map_y =[]
            color=[]
            for j in range(28):
                h_map_x= h_map_x + [j]*28
                h_map_y=h_map_y + [l for l in range(28)]
                color= color + [str(1-Images_l[m*int(len(Images_l)/plot_rows)+i][784-((28-j)+k*28)]/255) for k in range(28)]

            axs[m,i].scatter(h_map_x, h_map_y, s = 50, c = color)
    for ax in axs.flat:
        ax.label_outer()
    plt.savefig('Image_plot_'+ str(time.time()) + '.png')
    plt.show()
    return 0
```

Listing 3: Image Plot

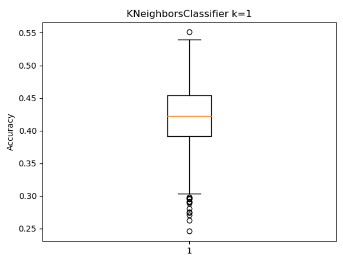
4.2 Boxplots



(a) SVC, kernel=rbf



(b) SVC, ker=poly, deg=2



(c) kNN, k=1

Figure 10: Boxplots of Accuracy

References

[1] D. Beneschi. *MNIST - EDA, Preprocessing & Classifiers.* URL: https://www.kaggle.com/damienbeneschi/mnist-eda-preprocessing-classifiers.

[2] B. E. Boser, I. M. Guyon, and V. N. Vapnik. *A Training Algorithm for Optimal Margin Classifiers.* COLT '92. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 1992, pp. 144–152. ISBN: 089791497X. URL: https://doi.org/10.1145/130385.130401.

[3] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge University Press, 2000. ISBN: 9781139643634. URL: https://books.google.at/books?id=I%5C_0gAwAAQBAJ.

[4] scikit-learn developers. *Nearest Neighbors.* Accessed: 2021-05-31. URL: https://scikit-learn.org/stable/modules/neighbors.html#nearest-neighbors-classification.

[5] T. Dietterich. "Overfitting and Undercomputing in Machine Learning". In: *ACM Comput. Surv.* 27.3 (Sept. 1995), pp. 326–327. ISSN: 0360-0300. URL: https://doi.org/10.1145/212094.212114.

[6] *Digit Recogniser.* Accessed: 2020-05-31. URL: https://www.kaggle.com/c/digit-recognizer/overview.

[7] S. Dye. "A primer on kernel methods". In: (2019). URL: %5Curl%7Bhttps://towardsdatascience.com/an-intro-to-kernels-9ff6c6a6a8dc%7D.

[8] E. Fix and J. L. Hodges. "Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties". In: (1951).

[9] G. James and D. Witten. *An Introduction to Statistical Learning.* Springer Science+Business Media New York, 2017. ISBN: 978-1-4614-7137-0.

[10] G. James and D. Witten. *An Introduction to Statistical Learning.* Springer Science+Business Media New York, 2017, pp. 29–33. ISBN: 978-1-4614-7137-0.

[11] S. B. Kotsiantis, D. Kanellopoulos, and P. E. Pintelas. "Data preprocessing for supervised leaning". In: *International Journal of Computer Science* 1.2 (2006), pp. 111–117. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.8413&rep=rep1&type=pdf.

[12] Y. LeCun, C. Cortes, and C. Burges. "MNIST handwritten digit database". In: *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist/* (2010).

[13] J. P. Marques de Sá. “Statistical Classification”. In: *Applied Statistics Using SPSS, STATISTICA and MATLAB*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. ISBN: 978-3-662-05804-6. URL: `https://doi.org/10.1007/978-3-662-05804-6_6`.

[14] P. Neuhart. *Linear Learning Machines*. University of Vienna, 2020.

[15] M. Nielsen. *Reduced MNIST: how well can machines learn from small data?* Accessed: 2021-04-03. URL: `http://cognitivemedium.com/rmnist`.

[16] *Pillow library*. Accessed: 2020-05-31. URL: `https://pillow.readthedocs.io/en/stable/`.

[17] *Recent Advances of Large-Scale Linear Classification*. Accessed: 2020-05-31. URL: `https://dmkd.cs.vt.edu/TUTORIAL/Bigdata/Papers/IEEE12.pdf`.

[18] *sklearn library*. URL: `https://scikit-learn.org/stable/`.

[19] D. Wilimitis. “The Kernel Trick in Support Vector Classification”. In: (2018). URL: `https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f`.