add to large margin intuition section that larger margin also reduces the space of possible hyperplanes use jabref for refs figure out where we need a vector space and where we need hilbert space

# Learning with Kernels

*Proseminar Data Mining*

Matthieu Bulté
Fakultät für Mathematik
Technische Universität München
Email: matthieu.bulte@tum.de

***Abstract*— Text der Kurzfassung**
***Index Terms*— support vector machines, machine learning, statistics**

## I. Introduction

todo

## II. Margin maximization classifier

The hard margin classifier learns the parameters $\mathbf{w} \in \mathbb{R}^n$ and $b \in \mathbb{R}$ of a hyperplane $\mathscr{H} \subset \mathbb{R}^n$, separating two classes of observations $\mathbf{x_i} \in \mathcal{X} \subset \mathbb{R}^n$. To train it, the algorithm takes a set of $m$ observations $\mathbf{x_i}$ together with a vector of labels $y_i = \pm 1$ for each of the observations:

$$(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \dots, (\mathbf{x_m}, y_m)$$

The learned parameters are then used to define a decision function $f$, assigning to points of $\mathcal{X}$ a label $\pm 1$ corresponding to the side of the hyperplane on which the points stands:

$$f(\mathbf{x}) = sgn(\langle \mathbf{w}, \mathbf{x} \rangle + b) \tag{1}$$

### A. Maximizating the margin

One particularity about the learning algorithm used for Support Vector Machines is the loss function the algorithm tries to optimize. Other learning algorithms typically chose a loss function based on the empirical risk, defined as following for a function $f$ and a training set $(\mathbf{x}, \mathbf{y})$ of size $m$:

$$R_{\text{emp}}[f] = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} |f(\mathbf{x_i}) - y_i| \tag{2}$$

The margin maximization algorithm instead, searches within the space of hyperplanes properly classifying the training set, for the hyperplane with the largest distance from the training points to the hyperplane.

The choice of maximizing the margin has good theoretical roots. Indeed, it is possible to derive an upper bound on the expected missclassification rate of a linear classifier based on the margin of the underlying hyperplane to the training set. It is beyond the scope of this work to go into further details but we will still attempt to provide an intuition on why it might be interesting to optimize it, and for more information the reader can refer to ref?.

The intuition comes from the fact that the observations from the test set, or encountered while using the classifier, have been generated by a similar process as the observations present in the training set. Thus, one can assume that observations are very similar to each other up to some noise. Maximizing the hyperplane margin $M$ is thus the same as increasing the tolerance to noise of the classifier. As shown in figure X, given a observation $x_i$ from the training set, all new obervations lying in the hypersphere of radius of the margin while be classified similarly to $x_i$. todo, add the figure

This idea translates well into the following optimization problem, trying to maximize the margin, while adding a constrain to force the classifier to propery classify the training set

$$\begin{aligned} \underset{\mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R}}{\text{maximize}} \quad & M := \frac{1}{\|\mathbf{w}\|} \min_i \, y_i(\langle \mathbf{w}, \mathbf{x_i} \rangle + b) \\ \text{subject to} \quad & y_i(\langle \mathbf{w}, \mathbf{x_i} \rangle + b) \geq M \text{ for all } i = 1 \dots m \end{aligned} \tag{3}$$

Assuming that a solution to this optimization problem exisits, it still has the issue that although the solution hyperplane is unique, there is an inifity of parameter vectors $\|\mathbf{w}\|$ resulting in the solution hyperplane, only differing in their length. Uniqueness of $\mathbf{w}$ can be ensured by adding to its length the following constraint:

$$M\|\mathbf{w}\| = 1$$

We can thus reformulate the optimization problem into the following quadratic optimization problem which is known to have a numerical solution:

$$\begin{aligned} \underset{\mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R}}{\text{minimize}} \quad & \|\mathbf{w}\| \\ \text{subject to} \quad & y_i(\langle \mathbf{w}, \mathbf{x_i} \rangle + b) \geq 1 \text{ for all } i = 1 \dots m \end{aligned} \tag{4}$$

### B. Support vectors

Although the previous formulation of the problem helped us to better understand the ideas of support vector machines, the resulting quadratic problem will become impractical when later introducing kernels methods. We will introduce in this section an equivalent formulation of the optimization problem (4)can latex handle references instead of hard coding? that will not only solve computational issues, but will also give us more insights about the resulting solution.

In order to get to these benefits, we introduce the Lagrangian $L$ together with Lagrange multipliers $\alpha_i \geq 0$:

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i [y_i(\langle \mathbf{w}, \mathbf{x_i} \rangle + b)) - 1] \quad (5)$$

This is called the dual formulation of our original optimization problem, called the primal problem, in which $\mathbf{w}$ and $b$ are called primal variables, and $\boldsymbol{\alpha}$ the dual variable. It can be shown ref? that the promal optimization problem is equivalent to finding a saddle point of $L$, minimizing with respect to $\mathbf{w}$ and $b$, while maximizing it with respect to $\boldsymbol{\alpha}$.

Because the solution of this dual problem is a saddle point, thus an extremum with respect to all variables. Thus our initial constrains are equivalent to the following constrains

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0 \text{ and } \frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0 \quad (6)$$

leading to

$$\sum_{i=1}^{m} \boldsymbol{\alpha}_i y_i = 0 \text{ and } \mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x_i} \quad (7)$$

Further, by replacing (7) and (8) in the objective function (6), we obtain an optimization problem free of primal variables, and end up only obtimizing against the vector $\boldsymbol{\alpha}$

$$\begin{aligned} \underset{\boldsymbol{\alpha} \in \mathbb{R}^n}{\text{maximize}} \quad & W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \boldsymbol{\alpha}_i - \frac{1}{2} \sum_{i,j=1}^{m} \boldsymbol{\alpha}_i \boldsymbol{\alpha}_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\ \text{subject to} \quad & \boldsymbol{\alpha} \geq 0 \text{ for all } i = 1 \dots m \\ & \sum_{i=1}^{m} \boldsymbol{\alpha}_i y_i = 0 \end{aligned} \quad (8)$$

Finally, the Karush-Kuhn-Tucker theorem states that the solution $\boldsymbol{\alpha}$ satisfies the following equality for all $i = 1 \dots m$ ref?

$$\boldsymbol{\alpha}_i [y_i(\langle \mathbf{w}, \mathbf{x_i} \rangle + b) - 1] = 0 \quad (9)$$

The importance of this equality is twofold. First, it allows us to compute $b$ from an observation $i$ for whichs $\boldsymbol{\alpha}_i \neq 0$. Second and most importantly, this equality implies that every observations not lying on the margin must have a vanishing Lagrangian coefficient. These special observations are called *support vectors*.

Support vectors have a special role. Indeed, looking at (7), one notices that the hyperplane only is only dependent on the support vectors. All other observations vanish from the sum because of their null Lagrange coefficient. This means that all the information extracted from the training set was represented by the support vectors, and all other observations had no role in the training of the algorithm, thus have no role in defining the decision boundary between the two classes of observations. This also means that. This statement can also be visualized in figure X where the algorithm learned a non linear decision boundary, in which one can observe how the support vectors define the path along which the decision boundary goes.

### C. Towards SVMs

We have chose in this work to present a simplified version of support vector machines called the hard margin classifier. This was done in order to focus on building an intuition of the way support vectors machines work. In practice, several considerations have to be made in order to make this classifier usable. These points will be handled in the next paragraphs.

The most important addition to the margin maximization algorithm that we have omited so far is the ability to train on a non linearly separable dataset caused by noise is the observations. Indeed, real world datasets are often generated from measuring physical phenomena. This kind of dataset is subject to noise that can cause an overlapping of the two classes around the decision boundary. Thankfully, these issues can be mitigated by introducing slack variables that will compensate the offset of an observation in the direction of the other class. Figure X illustrate the concept of these slack variables. A thorough investigation of this method can be found in ref?.

## III. Kernel methods

The previous section introduced the large margin classifier. The large margin classifier, as we saw, allows us the find the best seperating hyperplane between two linearly seperable classes of a dataset. In this section we introduce the concept of kernels, allowing us to learn non linear decision boudaries, continuing with the Mercer theorem and kernel trick, results from functional analysis allowing us to use the newly introducd ideas of kernels with the previously introduces agorithm, with only light modifications.

### A. Non linearly separable data

Very often, the decision boundary one is trying to learn is in it's nature non-linear. One trick that is often used in order to use linear learning algorithms is to add one or several dimensions to the data points by applying a non linear function of the coordinates of the point, to then learn the separating hyperplane in the projected space. This method is known as features engineering, because we add engineered features to our data points.

This approach would be very simple to implement in our current algorithm. Let $\phi : \mathcal{X} \to \mathcal{V}$ be a non linear transformation from our original space $\mathcal{X}$ to some higer dimensional Hilbert space $\mathcal{V}$. We can now modify the previously introduced algorithm by replacing simply every scalar product $\langle \cdot, \cdot \rangle$ by the scalar product on the projected points in the $\mathcal{V}$ space, namely $\langle \phi(\cdot), \phi(\cdot) \rangle$.

In order to better understand kernels and the kernel trick, let's follow the example of polynomial projection of degree $d$, in which every point is projected to a vector containing every monomial of degree $d$. For instance, choosing $d = 2$ together with a 2 dimensional input space leads to the projection $\phi : \mathbb{R}^2 \to \mathbb{R}^4$ defined as

$$\phi(x_1, x_2) \mapsto (x_1^2, x_2^2, x_1 x_2, x_2 x_1)$$

This representation, while adding more separation capabilities has the problem that the image feature space grows at an exponential rate together with $d$, making the choice of a larger $d$ prohibitivly expensive in terms of space usage. Though, the margin maximization algorithm only uses the scalar product of the observations in the feature space, and the following equations show that the projection in the high dimensional space is not required to the computation of the scalar product

$$\begin{aligned}\langle \phi(x), \phi(x') \rangle &= [x]_1^2[x']_1^2 + [x]_2^2[x']_2^2 + 2[x]_1[x]_2[x']_1[x']_2 \\ &= \langle x, x' \rangle^2 \\ &=: k(x, x')\end{aligned}$$

We call $k(\cdot,\cdot) : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ the kernel representation of the scalar product in the space $\phi(\mathbb{R}^2)$. One can generalize this idea to any polynomial degree $d$ as shown in ref?, making the computation of the scalar product in the $d$-th monomial space as trivial as computing the scalar product in the original space.

### B. Kernel trick

We have seen in the previous example, that it is sometimes possible to find a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ with $k(x, x') = \langle \phi(x), \phi(x') \rangle$ for some projection $\mathcal{X} \to \mathcal{V}$. With these special functions, it is possible to then, only lightly modifying our original algorithm, to run our learning algorithm in another vector space without the computational cost of projecting our data in this other space. This is called the kernel trick.

We will now change our point of view, and try to define the properties defining the class of functions being the representation of a scalar product of the projection of its inputs in another vector space. This means, which properties must hold for $k$ in order for a $\phi$ to exist with the correspondance property. better name needed, also a definition would be nice

Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ with $k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{V}}$ for some projection $\mathcal{X} \to \mathcal{V}$, because $\langle \cdot, \cdot \rangle_{\mathcal{V}}$ is a scalar product, the following properties must hold

- *Symmetry*

$$\begin{aligned}&\forall y_1, y_2 \in \mathcal{V}.\ \langle y_1, y_2 \rangle_{\mathcal{V}} = \overline{\langle y_2, y_1 \rangle_{\mathcal{V}}} && \Rightarrow \\ &\forall y_1, y_2 \in \phi(\mathcal{X}).\ \langle y_1, y_2 \rangle_{\mathcal{V}} = \overline{\langle y_2, y_1 \rangle_{\mathcal{V}}} && \Rightarrow \\ &\forall x_1, x_2 \in \mathcal{X}.\ \langle \phi(x_1), \phi(x_2) \rangle_{\mathcal{V}} = \overline{\langle \phi(x_2), \phi(x_1) \rangle_{\mathcal{V}}} && \Rightarrow \\ &\forall x_1, x_2 \in \mathcal{X}.\ k(x_1, x_2) = \overline{k(x_2, x_1)}\end{aligned}$$

- *Positive definitness*

$$\begin{aligned}&\forall y \in \mathcal{V}.\ \langle y, y \rangle_{\mathcal{V}} \geq 0 && \Rightarrow \\ &\forall y \in \phi(\mathcal{X}).\ \langle y, y \rangle_{\mathcal{V}} \geq 0 && \Rightarrow \\ &\forall x \in \mathcal{X}.\ \langle \phi(x), \phi(x) \rangle_{\mathcal{V}} \geq 0 && \Rightarrow \\ &\forall x \in \mathcal{X}.\ k(x, x) \geq 0\end{aligned}$$

We will show by construction, that these properties are sufficient to the proof of the existence of the desired space.

The literature contains several examples of vector spaces and projections with the desired property. We will here proove the existence of such a space by constructing the Reproducing Kernel Hilbert Space and a projection into it.

While being an approachable and sufficient prove of existence, the interested reader is invited to read about the Mercer Kernel Map, which is most often used to prove properties of the algorithm ref?.

maybe we want to introduce the idea that a scalar product is a way to measure similarity between vectors, thus kernels can be seen as a different way to measure similarity between observatiosn. Kernels contain domain knowledge and can enforce invariants

### C. Some useful kernels

## IV. Zusammenfassung und Ausblick

blabla