Learning with Kernels

Proseminar Data Mining

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

todo

- take care of refs
- 3d influence map is maybe a bit BS and could be replace by illustration of cosine similarity for text classification

***Abstract*— We present an algorithm for finding the observations relevant to classification of observations in two or more classes: the Support Vector Machine. The support observations are then used in the decision function as a linear combination, and their number allows us to derive an approximation of the classifier's performance. The kernel trick is then presented, a generic method appliable to inner product-based algorithm is shown to enable to let one incorporate domain knowledge about the classification problem in order to improve classification performance.**

***Index Terms*— support vector machines, kernels, machine learning, statistics**

I. Introduction

Machine learning can be seen as the science of automatically finding regularities or patterns in data. The contribution from many fields of science, engineering and mathematics brought a large variety of approaches to solving this problem.

In this work, we will bring our attention to the subject of kernel algorithms with a focus on Support Vector Machines (SVM). While most of the focus nowadays goes into neural networks, SVM stays a relevant algorithm with useful theoretical, but also practical properties that we will study.

The problem that we are trying to solve is the one of learning the decision function telling to which of two classes a point belongs for any point of the space points we are studying. More specifically, we are focusing on learning a linear classifier, a kind of classifier assigning a class to each point based on which side of a learned hyperplane the point lies, where a hyperplane is a generalization of a plane in three dimensions to an arbitrary high dimensional space.

The first section will focus on presenting on a simplified version of the SVM: the hard margin classifier. We will show how the algorithm constructs the optimal hyperplane for separating two classes of observations. This will help construct the necessary intuition to understand classical extensions to the algorithm as well as the kernelization of the algorithm.

In the second second, we will study kernel methods and the kernel trick. A method for working with a projection of the data in a higher dimensional space without having to pay the computational cost of applying the projection. We will concluse the section with an overview of different kernels, an discuss how kernels helps us to incroporate domain knowledge to the learned classifier.

We will then conclude this paper with an overview of practical challenges that can be found while using SVMs on modern data sets, and mention interesting extensions to support vector machines that were added in the 50 years of their existence.

II. Margin maximization classifier

The hard margin classifier learns the parameters $w \in \mathbb{R}^n$ 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:

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

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

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

A. Maximizating the margin

One particularity about the learning algorithm used for SVMs is the objective 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 1, 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.

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

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

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 $\|w\|$ resulting in the solution hyperplane, only differing in their length. Uniqueness of w can be ensured by adding to its length the following constraint:

$$M\|w\| = 1$$

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

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

B. Support vectors

Although the previous formulation of the problem helped us to better understand the ideas of the linear SVMs, 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) 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 $\alpha_i \geq 0$:

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

This is called the dual formulation of our original optimization problem, called the primal problem, in which 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 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

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

leading to

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

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 $\boldsymbol{\alpha}$

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

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

$$\boldsymbol{\alpha}_i[y_i(w\cdot \mathbf{x_i} + b) - 1] = 0 \tag{9}$$

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. Thus the new decision function can be rewritten as following

$$f(\mathbf{x}) = sgn\left(\sum_{\mathbf{x_i} \text{ is SV}} \alpha_i y_i \mathbf{x}\cdot\mathbf{x_i} + b\right) \tag{10}$$

Furthermore, it can be shown that the amount support vectors give us an upper bound on the out of sample error of the classifier. This means that a low number of support vectors can is an indicator of the capacity of a classifier to generalize to unseen samples. The upper bound is following

$$E\left[Pr\left(\text{error}\right)\right] \leq \frac{E\left[\text{number of support vectors}\right]}{\text{number of training observations}} \tag{11}$$

This inequality allows us to quickly derive a simple estimation of the quality of the learned classifier by just looking at the number of support vector.

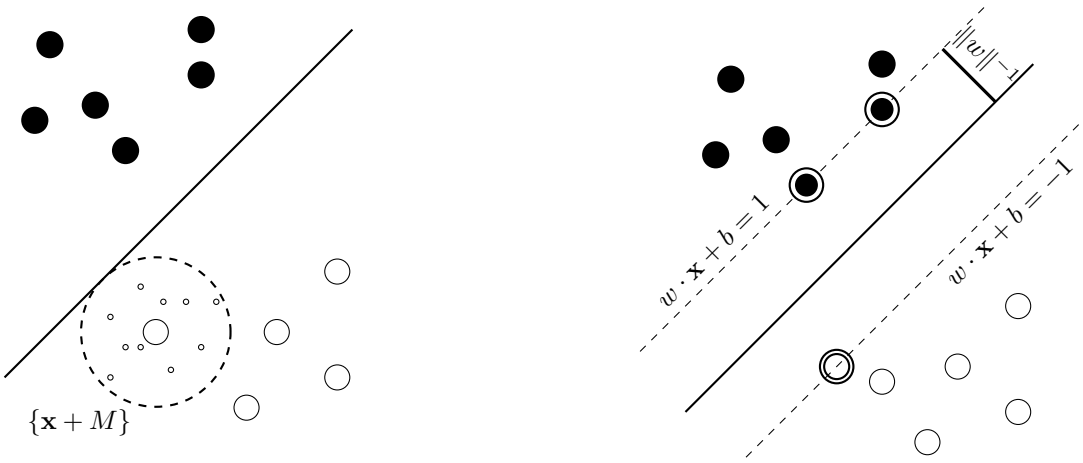

Fig. 1. Left side illustrates the idea behind margin maximization. Smaller points in the $\{\mathbf{x}+M\}$ are generated by adding noise to an observation of the training set. The right figure shows a dataset separated by a hyperplane with parameters w, b. The norm of w is determined by the distance from the plane to the support vectors.

C. Towards SVMs

We have chose in this work to present a simplified version of the linear SVMs called the hard margin classifier. This was done in order to focus on building an intuition of the way linear SVMs 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. A thorough investigation of this method can be found in ref?.

III. Kernel methods

The previous section introduced the linear SVM classifier. The linear SVM, 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 projhected 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 pre Hilbert space $\mathcal{V}$. A pre Hilbert space is a generalization of a Euclidean space, the geometrical space we are used to. Saying that the image of the projection should be a pre Hilbert space simply means that the image set is equiped with a scalar product, needed for our training algorithm. We can now modify the previously introduced algorithm by replacing simply every dot product by the scalar product on the projected points in the $\mathcal{V}$ space, namely $\langle\phi(\cdot), \phi(\cdot)\rangle_{\mathcal{V}}$.

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

$$\phi(x_1, x_2) \mapsto (x_1^2, x_2^2, x_1x_2, x_2x_1)$$

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, which is simply the dot product of the mapped vectors. The following equations show that the projection in the high dimensional space is not required to the computation of the scalar product

$$\begin{aligned}\phi(x)\cdot\phi(\tilde{x}) &= x_1^2\tilde{x}_1^2 + x_2^2\tilde{x}_2^2 + 2x_1x_2\tilde{x}_1\tilde{x}_2\\ &= (x\cdot\tilde{x})^2\\ &=: k(x,\tilde{x})\end{aligned}$$

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

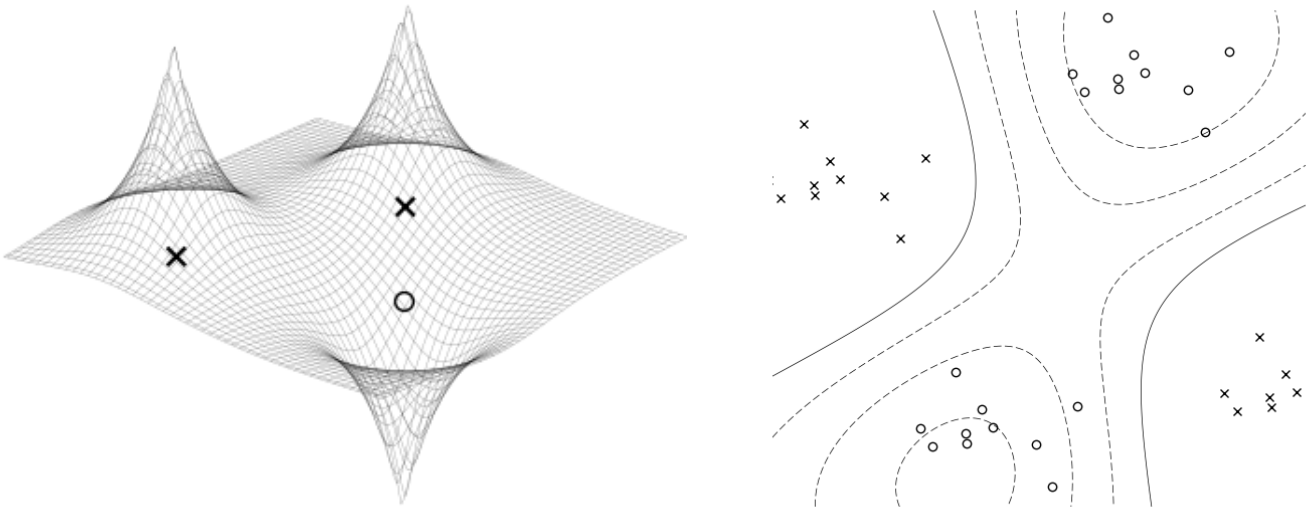

Fig. 2. Left side shows on the vertical axis the influence level of the point on the 2d plane from two positive points drawn with **x** markers and of one negative points marked with **o**. On the right pane, the solid line reprensents the decision boundary and the dashed lines represents the contour levels of the decision function with learned using an RBF kernel.

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 $k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{V}}$ for some projection $\phi : \mathcal{X} \to \mathcal{V}$. With these special functions, it is possible by only lightly modifying our original algorithm to run the learning algorithm in another pre Hilbert 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 ϕ to exist with the correspondance property.

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* Positive definitness of a kernel is a stronger property than the one required for the positive definiteness in the features space. A kernel is said to be positive definite if for every $\alpha_1, ..., \alpha_n \in \mathbb{R}$ following inequality holds

$$\sum_{i,j=1}^{n} \alpha_i \alpha_j k(x_i, x_j) \geq 0$$

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 construction.

Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be symmetrical and positive definite, and $\mathbb{R}^{\mathcal{X}}$ be the set of functions from a non empty set $\mathcal{X}$ to $\mathbb{R}$. We define the reproducing kernel map as follow

$$\begin{aligned}
\phi : \mathcal{X} &\to \mathcal{H} \\
\mathbf{x} &\mapsto k(\cdot, \mathbf{x})
\end{aligned} \tag{12}$$

We now show how the mapped observations of the training set $\{k(\cdot, \mathbf{x}_1), ..., k(\cdot, \mathbf{x}_n)\}$ spans a Hilbert space in which the reproducing property hold with the scalar defined on the span set $\langle k(\cdot, \mathbf{x}_i), k(\cdot, \mathbf{x}_j) \rangle = k(\mathbf{x}_i, \mathbf{x}_j)$ and its canonical derivation for the spanned space

$$\begin{aligned}
\langle f, g \rangle &= \langle \sum_{i=1}^{n} \alpha_i k(\cdot, \mathbf{x}_i), \sum_{j=1}^{n} \beta_j k(\cdot, \mathbf{x}_j) \rangle \\
&= \sum_{i,j=1}^{n} \alpha_i \beta_j \langle k(\cdot, \mathbf{x}_i), k(\cdot, \mathbf{x}_j) \rangle \\
&= \sum_{i,j=1}^{n} \alpha_i \beta_j k(\mathbf{x}_i, \mathbf{x}_j)
\end{aligned}$$

Positive definitness, bilinearity and symmetry can all be trivially derived from the definition of this scalar product together with the symmetry and positive definitness of the kernel function. Then attentive reader will notive that the functions $\{k(\cdot, x_1), ..., k(\cdot, x_n)\}$ must nost necessarily be linearly independant, leading to a non unqiue mapping from f to the coefficients $\alpha_1, ..., \alpha_n$. To see that this doesn't imply the ill definitness of the defined scalar product, we note that

$$\langle f, g \rangle = \sum_{j=1}^{n} \beta_j f(\mathbf{x}_j)$$

and by symmetry

$$\langle f, g \rangle = \langle g, f \rangle = \sum_{i=1}^{n} \alpha_i g(\mathbf{x}_i)$$

Which shows that the scalar product does not depend on the choice of coefficients for the functions f and g confirming the scalar product is well defined. Thus, the previously defined scalar product, together with the norm $|f| = \sqrt{\langle f, f \rangle}$ define a valid Hilbert space and the correspondance property holds

$$\begin{aligned} \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle &= \langle k(\cdot, \mathbf{x}_i), k(\cdot, \mathbf{x}_j) \rangle \\ &= k(\mathbf{x}_i, \mathbf{x}_j) \end{aligned}$$

Now that we have presented the kernel trick as well as the existence of a corresponding Hilbert space, the next section will focus on finding and creating useful kernels, as well as presenting some classical kernels.

C. Some useful kernels

We have defined in the last section what kernels are and how we can modify our algorithm to learn non-linear boundaries in our data. What we have omited so far is how is one supposed to choose develop or choose a kernel when facing a practical problem.

One first way to choose a kernel is by using existing knowledge about the shape the decision boundary should have. Let's recall the equation of the decision boundary defined by the learned parameters using the kernel k

$$\mathscr{H} = \left\{ \mathbf{x} \in \mathbb{R}^n \mid \sum_{\mathbf{x}_i \text{ is support vector}} \alpha_i y_i k(\mathbf{x}, \mathbf{x}_i) + b = 0 \right\}$$

For illustration, if we choose the kernel to be a polynomial kernel of degree 2, meaning $k(\mathbf{x}, \tilde{\mathbf{x}}) = (\mathbf{x} \cdot \tilde{\mathbf{x}} + 1)^2$, we can refine the definition of the decision boundary as the set of every $\mathbf{x} \in \mathbb{R}^2$ for which following holds

$$\begin{aligned} 0 = &\ (w \cdot \mathbf{x} + 1)^2 + b \\ = &\ (w_1 x_1 + w_2 x_2 + 1)^2 + b \\ = &\ (w_1 x_1)^2 + (w_2 x_2)^2 + 2 w_1 x_1 w_2 x_2 + w_1 x_1 + w_2 x_2 + 1 + b \\ = &\ \mathbf{x}^T \begin{bmatrix} w_1^2 & w_1 w_2 \\ w_1 w_2 & w_2^2 \end{bmatrix} \mathbf{x} + w \cdot \mathbf{x} + 1 \end{aligned}$$

The decision boundary formed by this equation is called a quadric, the generalization of a conic section, which takes a shape from a known set of different kind of solutions. Thus, if one knows a quadric decision boundary is needed, then using a polynomial kernel of degree 2 will learn the proper parameters of the quadric.

We have seen how previous knowledge on the shape of the decision boundary can lead to the choice of a particular kernel. Unfortunately, it is often the case that the best kind of decision boundary can't be determined. Thankfully, there is still a way to use domain knowledge about the problem to choose a kernel that will best help training a performant classifier.

In order to incorporate this other kind of domain knowledge, we have to look at kernels, not as the computations of a scalar product in some feature space, but as a similarity measurement. Taking this point of view, the kernel trick becomes a way to find the proper Hilbert space in which one's definition of similarity defines a scalar proper product, letting us use the Support Vector Machine machinery as a way to train a classifier based on our definition of similarity.

What do we mean by similarity? Let's first write down a more geometrical definition of the euclidean scalar product in $\mathbb{R}^n$, the inner product

$$\mathbf{x} \cdot \tilde{\mathbf{x}} = \|\mathbf{x}\| \|\tilde{\mathbf{x}}\| \cos(angle(\mathbf{x}, \tilde{\mathbf{x}}))$$

This equivalent definition of the euclidean scalar product makes it more visible what the underlying notion of similarity of the euclidean scalar product is measured by the angle between the two vectors, scaled by their length. A similar kernel often used is the cosine similarity kernel, defined as

$$k(\mathbf{x}, \tilde{\mathbf{x}}) = \frac{\mathbf{x} \cdot \tilde{\mathbf{x}}}{\|\mathbf{x}\| \|\tilde{\mathbf{x}}\|} = \cos(angle(\mathbf{x}, \tilde{\mathbf{x}}))$$

This notion of similarity will thus compare observations by first normalizing them to then compare the directions of the vectors. This similarity measure is often used in the context of text classification, where each entry of the vector corresponds to the number of occurences of a word in a text. The normalization will transform count of occurences in frequencies, thus having a similarity based on the relative importance of each word, independantly of the length of the text.

Cosine similarity is an interesting similarity, but one natural way to reason about similarity of points is their distance from one another. The next kernel we introduce incorporates the notions of distance as a similarity measurement, is called the Radial Basis Function kernel and is defined as followed for some hyperparameter $\sigma \in \mathbb{R}$

$$k(\mathbf{x}, \tilde{\mathbf{x}}) = \exp\left(-\frac{\|\mathbf{x} - \tilde{\mathbf{x}}\|^2}{\sigma^2} \right)$$

This kernel contains the idea that similarity between two observations should decay exponentially with the distance between the two vectors. This kernel isn't only a good model

of our notion of similarity, but it also has interesting properties when used with the Support Vector Machines training algorithm.

Let's first recall what was the decision function that was learned by the algorithm

$$f(x) = sgn\left(\sum_{\mathbf{x}_i \text{ is support vector}} \alpha_i y_i k(\mathbf{x}, \mathbf{x}_i) + b\right)$$

We evaluate the sign of a linear combination of the similarity of the current observation to the support vectors. One can see understand each support vector as a point of influence in the direction of the class it belongs to, with it's influence intensity exponentially decaying with the distance. The observation will then be classified by taking a weighted sum of the influence of each support vector and verifying if it is over or under some threshold b. This intuition can be visualised in figure 2 where we show the influence map built by two points on in a one dimensional space as well as a decision boundary learned using a radial basis function kernel.

IV. Conclusion

In this paper we have shown following

- Changing the objetive function of the learning problem to maximizing the margin from the hyperplane to the training set not only result in a theoretically better classifier, but also brought us more information about the nature of the classification problem through the support vectors.
- There exists a sound theory allowing us to change any scalar product based learning algorithm to use a kernelized version, allowing to run the algorithm in a non linear projection of the training set, without having to actually run the projection.
- Kernels are not only a way to improve performances of the training and classification, but they allow us to incorporate domain knowledge to the classifier in order to improve the performance of the classifier.

We have seen the good sides of Support Vector Machines, omiting sometimes to mention some of the downsides of this algorithm. The two main issues one finds with Support Vector Machine is the high cost of the optimization problem being solved. Also, a fair amount of time has to be spent in order to find the proper combination of hyperparameters such as the choice of the kernel but also its parameters, such as the degree of the polynomial kernel.

Thankfully, these issues can be tackled with modern tooling. Modern implementations of Support Vector Machines solve the performance issues for some specific cases, such as the pegasos algorithm ref? for training a linear classifier. Concerning the choice of parameters, one can use methods such as cross validated grid search in order to find combinations of hyper parameters that perform best without overfitting the training set.

Thus, even in the days of deep neural networks, Support Vector Machine stays a relevant learning algorithm. Their comparatively good training performances as well as their low number of hyper parameters tuning and the generalization capacity of the learned classifier makes Support Vector Machines an easy to use algorithm that also provides satisfying resulsts.