Learning with Kernels

Proseminar Data Mining

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

***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 applicable 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. Contributions 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 focusing on Support Vector Machines (SVM). While most of the effort 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 to 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 present 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 data points, or observations as we will call them in this work. This will help construct the necessary intuition to understand classical extensions to the algorithm.

In the second section, 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 finish with an overview of different kernels, and discuss how kernels help us incorporate 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 $b \in \mathbb{R}$ defining the hyperplane $\mathscr{H} := \{\mathbf{x} \in \mathbb{R}^n \mid w \cdot \mathbf{x} + b = 0\}$, separating two classes of observations $\mathbf{x_i} \in \mathbb{R}^n$. To train this classifier, 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 indicating which class each observation belongs to.

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)$$

The learned parameters are then used to define a decision function f, assigning a label ± 1 to points of $\mathbb{R}^n$ corresponding to the side of the hyperplane on which the observation stands

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

A. Maximizing the margin

One particularity about the SVM's learning algorithm is the objective function it tries to optimize. Other learning algorithms typically choose a loss function based on the empirical risk, defined for a classifier f and a training set $(\mathbf{x}, \mathbf{y})$ of size m as following

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

The margin maximization algorithm instead searches within the space of hyperplanes properly classifying the training set for the hyperplane with the largest margin: 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 misclassification 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 understand why maximizing the margin can lead to better generalization. For more information the reader can refer to [1].

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

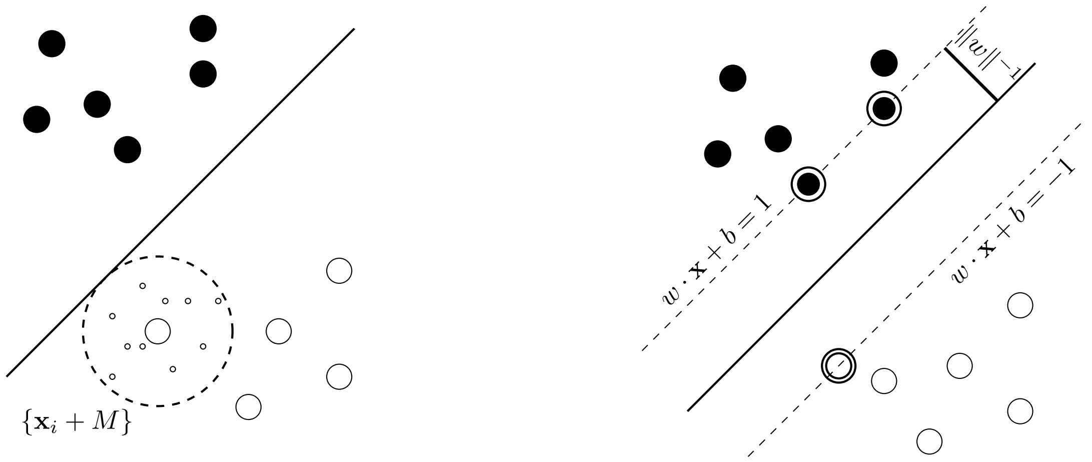

Fig. 1. Left side illustrates the idea behind margin maximization. Smaller points in the $\{\mathbf{x} + M\}$ are generated by adding noise to an observation of the training set. The right figure shows a dataset separated by a hyperplane with parameters w, b. The norm of w is determined by the distance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

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 thereby equivalent to increasing the classifier's tolerance to noise. As shown in figure 1, given an observation $\mathbf{x}_i$ from the training set, all new observations lying within the hypersphere of radius of the margin will be classified similarly to $\mathbf{x}_i$.

This idea translates well into the following optimization problem, trying to maximize the margin while constraining the search space to the set of classifiers properly classifying 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 exists, it still has the issue that although the solution hyperplane is unique, there is an infinity of parameter vectors w resulting in the solution hyperplane, only differing in their length. Uniqueness of the result might seem threatened, but as the w only expresses the direction of the hyperplane and its margin, uniqueness can be ensured by adding the following constraint to its length

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

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 optimization 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 now reformulate an equvalent optimization problem (4) that will not only solve the computational issues, but will also give us more insights about the resulting solution.

In order to get to these benefits, we introduce the Lagrangian function L together with Lagrange multipliers $\boldsymbol{\alpha}_i \geq 0$

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

This is called the dual formulation of the original optimization problem, called the primal problem, in which w and b are called primal variables and $\boldsymbol{\alpha}$ the dual variable. The primal optimization problem is equivalent to finding a saddle point of L, minimizing with respect to w and b, while maximizing with respect to $\boldsymbol{\alpha}$, this is called the Wolf dual [2].

The solution to this problem is a saddle point, a different kind of extremum with respect to each of the variables. Because the solution is an extremum, we know that the partial derivative with respect to each of the variables will be zero. Thus the following equalities must hold

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

Which simplifies to the following constraints

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

By replacing (6) and (7) in the Lagrangian (5), we obtain an optimization problem free of primal variables, and only optimize 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 condition [3], from optimization theory, states that the solution $\boldsymbol{\alpha}$ satisfies the following equality for all $i = 1\dots m$

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

The importance of this equality is twofold. Firstly, it allows us from one observation $\mathbf{x}_j$ and its label y_j for which the Lagrangian multiplier $\boldsymbol{\alpha}_j \neq 0$ to compute b as following

$$\begin{aligned}\boldsymbol{\alpha}_j[y_j(w\cdot\mathbf{x}_j + b) - 1] = 0 \\ \Rightarrow b = \frac{1}{y_j} - w\cdot\mathbf{x}_j\end{aligned} \tag{10}$$

Secondly, this equality implies that every observation not lying on the margin must have a vanishing Lagrangian coefficient, meaning $\boldsymbol{\alpha}_i = 0$. These special observations are called *support vectors*. As we have shown in (7) and (10), the parameters w and b that uniquely define the separating hyperplane can both be expressed only in terms of the support vectors. This means that support vectors carry all the information that was to be extracted from the training set in order to create the decision boundary. The decision function can be rewritten as following

$$f(\mathbf{x}) = sgn\left(\sum_{\mathbf{x}_i \text{ is SV}} \boldsymbol{\alpha}_i y_i \mathbf{x}\cdot\mathbf{x}_i + b\right) \tag{11}$$

Furthermore, it can be shown [1] that the number of support vectors give us the following upper bound on the classifier's out of sample error. This means that a low number of support vectors can is an indicator of the capacity of a classifier to generalize to unseen samples.

$$\mathbb{E}\left[\Pr(\text{error})\right] \leq \frac{\mathbb{E}\left[\text{number of support vectors}\right]}{\text{number of training observations}} \tag{12}$$

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.

C. Towards SVMs

In this work, we have chosen to present a simplified version of the linear SVM 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.

The most important addition to the margin maximization algorithm that we have omitted so far is the ability to train on a data set where the constraint in our optimizing problem cannot be fulfilled. 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, making it impossible to find a boundary properly separating the two classes of observations. 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 [4].

III. Kernel methods

The previous section defined the linear SVM classifier. As we have seen, the linear SVM allows us the find the best separating hyperplane between two linearly separable classes of observations. In this section we introduce the concept of kernels, allowing us to learn non linear decision boundaries. We continue with the *Reproducing Kernel Hilbert Space* and the *kernel trick*, results from functional analysis allowing us to use the newly introduced idea of kernels with the previously presented algorithm.

A. Non linearly separable data

Very often, the decision boundary one is trying to learn is of nature non-linear. One method that is often used to make a hyperplane learning algorithm learn a non-linear decision boundary is to first map the dataset to another space using a non linear transformation. The learned boundary is then linear in the mapped space, but because the chosen projection is not linear, the pre-image of the decision boundary must not be linear.

This approach is very simple to implement and only requires light modifications to the presented algorithm. Let $\phi : \mathbb{R}^n \to \mathcal{V}$ be our projection into the feature space $\mathcal{V}$. In order for the SVM algorithm to be usable, the $\mathcal{V}$ space must be a *pre-Hilbert space*.

A pre-Hilbert space is a vector space equipped with a scalar product. It is simply a generalization of the usual geometrical space, the Euclidian space. This generalization allows us to work with possibly infinitly dimensional spaces, such as for instance the vector space of absolutely converging sequences ℓ_1, the same way as we work with the more intuitive vector spaces. In this case, the optimization problem at the heart of the SVM's training algorithm is only based on the scalar product between mapped observations, thus requiring the feature space to be a pre-Hilbert space.

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 analyze 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 $\phi : \mathbb{R}^2 \to \mathbb{R}^3$ defined as

$$\phi(x_1, x_2) \mapsto (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

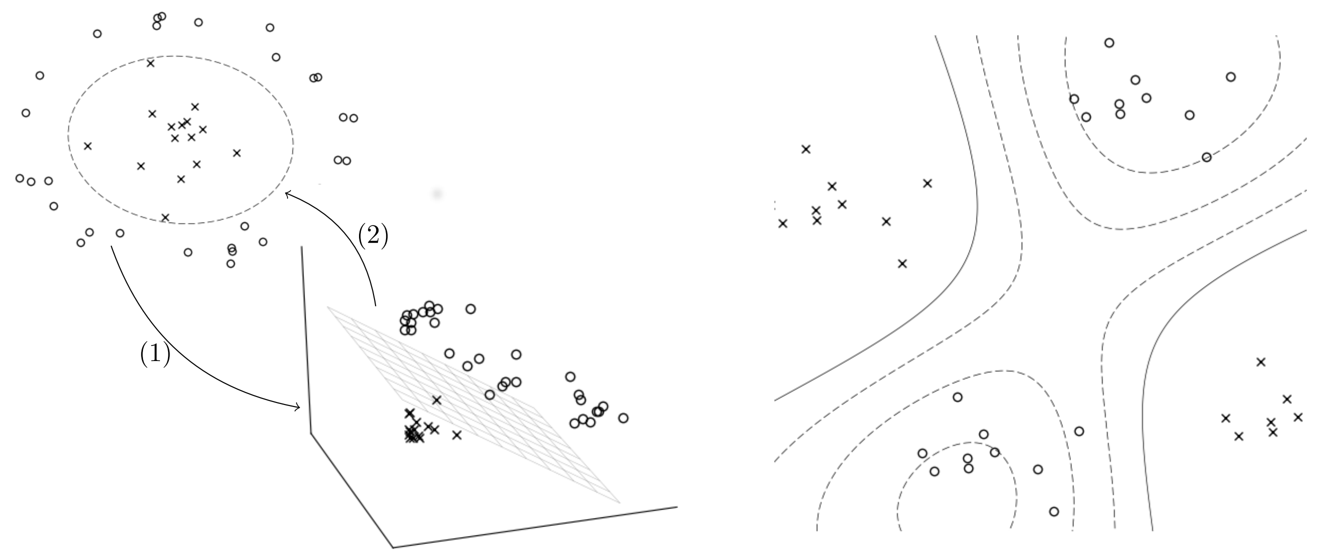

Fig. 2. Le left pane illustrates how a non linear projection in a higher dimensional space makes (1) it possible to learn non linear decisions boundaries in the observations space by projecting back the learned linear decision boundary (2). On the right pane, the solid line represents the decision boundary and the dashed lines represents the contour levels of the decision function with learned using an RBF kernel.

This projection, illustrated in figure 2, makes it possible to learn a decision boundary that is not linear in the observations but it has the problem that the dimension of the image space grows at an exponential rate together with d, making the choice of a larger d prohibitively expensive. 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 higher dimensional space is not required to the computation of the scalar product, and can be replaced by a function of the two vectors in the original space

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

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 show that such a function exists for any polynomial degree d, 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 $k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ with $k(x, x') = \langle\phi(x), \phi(x')\rangle_{\mathcal{V}}$ for some projection $\phi : \mathbb{R}^n \to \mathcal{V}$. Such special functions, it is possible by only lightly modifying the original algorithm to run the learning algorithm in another pre-Hilbert space without the computational cost of projecting the data in this other space. This is called the kernel trick.

We will now change our point of view and stop looking for a smart way to avoid projecting in the feature space, but instead try to define the set of functions k, called *kernel functions*, for which it is possible to construct a pre-Hilbert space $\mathcal{V}$ and a projection ϕ such that $k(x, x') = \langle\phi(x), \phi(x')\rangle_{\mathcal{V}}$. This, as we will show in the next section, will make it possible to express a large variety of decision boundaries through the choice of the kernel function without having to search for a corresponding pre-Hilbert space and a mapping ϕ.

Let $k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, k is said to be a *kernel function* if the following properties hold

- *Symmetry*

$$\forall x_1, x_2 \in \mathbb{R}^n.\ k(x_1, x_2) = k(x_2, x_1)$$

- *Positive definiteness*

$$\sum_{i,j=1}^{m} c_i c_j k(x_i, x_j) \geq 0$$

For any $x_1, ..., x_m \in \mathbb{R}^n$ and $c_1, ..., c_m \in \mathbb{R}$

We now show by construction that these properties are sufficient to the proof of the existence of the desired pre-Hilbert space. The literature contains several examples of vector spaces and projections with the desired property, but we will here prove the existence of such a space by constructing one of those.

Let $k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ be a kernel, and $\mathbb{R}^{\mathbb{R}^n}$ be the set of functions from $\mathbb{R}^n$ to $\mathbb{R}$. We define the reproducing kernel map

$$\begin{aligned} \phi : \mathbb{R}^n &\to \mathbb{R}^{\mathbb{R}^n} \\ \mathbf{x} &\mapsto k(\cdot, \mathbf{x}) \end{aligned} \tag{13}$$

Using this projection, the span of the mapped training observations constructs the following sub-vector space of the vector space of functions

$$\begin{aligned} \mathcal{V} &:= \text{span}\left(\{\phi(\mathbf{x}_1), ..., \phi(\mathbf{x}_m)\}\right) \\ &= \left\{ \sum_{i=1}^{m} \lambda_i k(\cdot, \mathbf{x}_i) \mid \lambda_1, ..., \lambda_m \in \mathbb{R} \right\} \end{aligned}$$

The constructed vector space is a subset of the infinite dimensional vector space of functions from $\mathbb{R}^n$ to $\mathbb{R}$, but because it was constructed as the span of finitely many elements, we know that the dimension of the spanned space is at most m. We can now define a scalar product on $\mathcal{V}$ by first defining it on its spanning elements $\langle k(\cdot, \mathbf{x}_i), k(\cdot, \mathbf{x}_j) \rangle_{\mathcal{V}} = k(\mathbf{x}_i, \mathbf{x}_j)$. The definition can be extended to the rest of the vector space, defining the scalar product between two arbitrary functions $f = \sum_{i=1}^{m} \alpha_i k(\cdot, \mathbf{x}_i)$ and $g = \sum_{i=1}^{m} \beta_i k(\cdot, \mathbf{x}_i)$ elements of $\mathcal{V}$ as

$$\begin{aligned} \langle f, g \rangle_{\mathcal{V}} &= \langle \sum_{i=1}^{n} \alpha_i k(\cdot, \mathbf{x}_i), \sum_{j=1}^{n} \beta_j k(\cdot, \mathbf{x}_j) \rangle_{\mathcal{V}} \\ &= \sum_{i,j=1}^{n} \alpha_i \beta_j \langle k(\cdot, \mathbf{x}_i), k(\cdot, \mathbf{x}_j) \rangle_{\mathcal{V}} \\ &= \sum_{i,j=1}^{n} \alpha_i \beta_j k(\mathbf{x}_i, \mathbf{x}_j) \end{aligned} \tag{14}$$

The properties required for $\langle \cdot, \cdot \rangle_{\mathcal{V}}$ to be a scalar product directly follow from its definition. Bilinearity is a direct consequence of the extension of the definition to the whole span of the basis functions. As for symmetry and positive definiteness, these were set as conditions for k to be a kernel function. We have constructed a pre-Hilbert space for which k is equivalent to the scalar product of projected points

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

Now that we have presented the kernel trick as well as the existence of a corresponding pre-Hilbert space for every kernel function. One can now treat the SVM algorithm as a black box in which it is possible to plug any kernel function without having to think about the existence of any pre-Hilbert space or mapping ϕ. The next section will study the process of 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 omitted 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

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

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

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

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

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

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, independently 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

$$f(x) = sgn\left(\sum_{\mathbf{x}_i \text{ is SV}} \boldsymbol{\alpha}_i y_i k(\mathbf{x}, \mathbf{x}_i) + b\right)$$

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.

We have shown with three examples how previous knowledge can help train high quality classifiers. Because we had to omit many interesting concepts of choice of kernel, the reader is invited to consult further resources to learn about more mechanical ways to construct kernels [5] but also about invariane incorporation [6] or advanced approach in kernel design for solving modern complex problems [7].

IV. Conclusion

In this paper we have shown following

- Changing the objective 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, omitting 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 [8] 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 results.

References

[1] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York, NY, USA: Springer-Verlag New York, Inc., 1995.

[2] R. Fletcher, *Practical Methods of Optimization; (2Nd Ed.)*. New York, NY, USA: Wiley-Interscience, 1987.

[3] H. W. Kuhn and A. W. Tucker, "Nonlinear programming," in *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*. Berkeley, Calif.: University of California Press, 1951, pp. 481–492.

[4] B. Schoelkopf, C. Burges, and V. Vapnik, "Extracting support data for a given task," in *Proceedings, First International Conference on Knowledge Discovery & Data Mining, Menlo Park*. AAAI Press, 1995, pp. 252–257.

[5] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001.

[6] D. Decoste and B. Schölkopf, "Training invariant support vector machines," *Machine Learning*, vol. 46, no. 1, pp. 161–190, 2002. [Online]. Available: http://dx.doi.org/10.1023/A:1012454411458

[7] F. Lauer, C. Y. Suen, and G. Bloch, "A trainable feature extractor for handwritten digit recognition," *Pattern Recognition*, vol. 40, no. 6, pp. 1816–1824, 2007, 19 pages. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00018426

[8] S. Shalev-Shwartz, Y. Singer, and N. Srebro, "Pegasos: Primal estimated sub-gradient solver for svm," in *Proceedings of the 24th International Conference on Machine Learning*, ser. ICML '07. New York, NY, USA: ACM, 2007, pp. 807–814. [Online]. Available: http://doi.acm.org/10.1145/1273496.1273598