Learning with Kernels

Harshita Agarwala
M.Sc in Mathematics in Data Science
Technische Universität München

March 9, 2020

Abstract

This paper describes the central ideas of Support Vector (SV) Classification. Its goal is to provide an overview of the basic concepts. It also gives an introduction to kernels and discusses their use in faster computation of different learning algorithms. It reviews the main kernel algorithms like SV classifiers and Kernel-Principal Component Analysis. It also provides insights from Statistical Learning Theory and presents important mathematical formulations on performance of functions learnt by these algorithms. Moreover, it provides results from a python implementation of the SV classification problem discussed in the paper.

Contents

1 Introduction

In this context, learning refers to the process of inferring general rules by observing examples. For instance, children can learn what "a car" is, just by being shown examples of objects that are cars and objects that are not cars. The field of machine learning studies the process of learning in the abstract. Machine learning has roots in artificial intelligence, statistics, and computer science, but by now has established itself as a scientific discipline in its own right.

There are two types of learning algorithms: supervised and unsupervised. Classification is an example of a supervised learning problem: the training examples consist both of instances(inputs) and correct labels. Therefore the goal is to find a functional relationship between them. On the other hand, the training data in the unsupervised setting only consists of instances (inputs), without any further information about what kind of output is expected on those instances. Here, the question of learning is more about learning a structure on the underlying space of instances. A standard example of such a setting is clustering. In this paper, we focus on a classification problem and then go to a form of unsupervised learning.

2 Data Representation and Similarity

A basic problem of learning theory is assigning a new object to a class given some empirical data or training data.

$$(x_1, y_1), ..., (x_m, y_m) \in \chi \times \{\pm 1\} \tag{1}$$

Here, χ is a nonempty set of patterns/inputs x_i and the y_i are called labels, targets, or outputs. This is called (binary) pattern recognition or (binary) classification. To study the problem of learning, an additional kind of structure is required that generalizes to unseen data points. In the case of pattern recognition, this means that given some new pattern $x \in \chi$, the model should be able to predict the corresponding $y \in \{\pm 1\}$.

Intuitively, such a y has to be chosen such that (x, y) is in some sense **similar** to the training examples in eq (1). Hence, notions of similarity in χ and $\{\pm 1\}$ are needed.

These similarity measures are also referred to as kernels [1]. Kernel functions are of the form:
$k : \chi \times \chi \to \mathbb{R}, (a, b) \to k(a, b)$

A simple kernel function is the Dot Product/Scalar Product: $< \mathbf{a}, \mathbf{b} >= \mathbf{b}^T\mathbf{a}$ where $\mathbf{a}$ and $\mathbf{b}$ are vectors. The geometric interpretation of the canonical dot product is that it computes the cosine of the angle between the vectors a and b provided they are normalized to unit length: $\cos(\mathbf{a}, \mathbf{b}) =< \mathbf{a}, \mathbf{b} > /(||\mathbf{a}||||\mathbf{b}||)$.

The norm or the length of a vector a is given by: $||\mathbf{a}|| = \sqrt{(< \mathbf{a}, \mathbf{a} >)}$. The patterns x can be represented in a dot product space H by a mapping $x \to \boldsymbol{x} = \phi(x)$, here $\boldsymbol{x}$ is a vector in the space H (also called Feature Space)

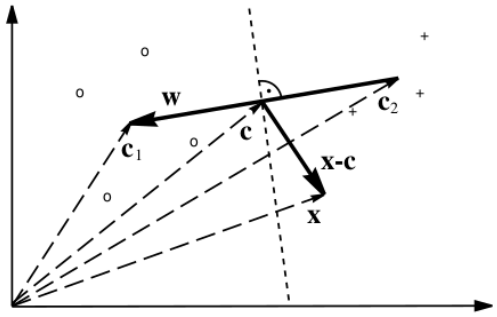

Figure 1: Simple Pattern Recognition Algorithm

3 Simple Pattern Recognition Algorithm

We are now in the position to describe a pattern recognition learning algorithm that is one of the simplest possible. We assume that the data is embedded into a dot product space H. Using the dot product, we can measure distances in that space. The basic idea of the algorithm will be to assign a previously unseen pattern to the class whose mean is closer.

We first begin by computing the mean of the two classes in feature space H:

$$c_1 = \frac{1}{m_1} \sum_{i:y_i=1} x_i \tag{2}$$

$$c_2 = \frac{1}{m_2} \sum_{i:y_i=-1} x_i \tag{3}$$

where m_1 and m_2 are the number of examples with positive and negative labels,respectively. We assume that $m_1, m_2 > 0$, that is there exists at least one data point of each label. We then assign a new point x to $+1$ if it is closer to c_1 and -1 if it is closer to c_2. From (Figure:1) we can see the geometric construction can be formulated in terms of the dot product $< ., . >$. Half-way in between c_1 and c_2 lies the point $c = (c_1 + c_2)/2$. We compute the class of x by checking whether the vector $x - c$ connecting c to x encloses an angle smaller than $\pi/2$ with the vector $w = c_1 - c_2$ connecting the class means. This leads to:

$$\begin{aligned} y &= sgn < (x - c), w > \\ &= sgn < x - (c_1 + c_2)/2, c_1 - c_2 > \\ &= sgn(< (x, c_1 > - < x, c_2 > + b) \end{aligned} \tag{4}$$

where b is defined as $b = \frac{1}{2}(||c_1||^2 + ||c_2||^2)$. It can be clearly seen from (Figure:1), that the equation (4) induces a decision boundary which has the form of a hyperplane.

We first express the equation (4) by replacing c_1 and c_2 with equations (2) and (3):

$$y = sgn(\frac{1}{m_1}\sum_{i:y=1} < x, x_i > + \frac{1}{m_2}\sum_{i:y=-1} < x, x_i > + b) \tag{5}$$

As discussed earlier, the dot products are a kernel. We can then replace them with the function $k : \chi \times \chi \to \mathbb{R}$, $(a, b) \to k(a, b)$. Hence, the final equations are:

$$y = sgn(\frac{1}{m_1}\sum_{i:y=1} k(x, x_i) + \frac{1}{m_2}\sum_{i:y=-1} k(x, x_i) + b) \tag{6}$$

and then the offset can be written as

$$b = \frac{1}{2}(\frac{1}{m_2}^2 \sum_{i:y=-1} k(x, x_i) - \frac{1}{m_1}^2 \sum_{i:y=1} k(x, x_i)\ ) \tag{7}$$

4 Introduction to Statistical Learning Theory

Statistical learning theory (SLT) is a theoretical branch of machine learning and attempts to lay the mathematical foundations for the field. [2] The questions asked by SLT are fundamental:

1. Which learning tasks can be performed by computers in general (positive and negative results)?
2. What kind of assumptions do we have to make such that machine learning can be successful?
3. What are the key properties a learning algorithm needs to satisfy in order to be successful?
4. Which performance guarantees can we give on the results of certain learning algorithms?

Let us first assume that the data is generated independently from some unknown (but fixed) probability distribution $P(x, y)$. This is usually a standard assumption in learning theory. The data generated this way is called independent and identically distributed (iid). Hence, now the goal is to find a function f that correctly classifies unseen examples (x', y'). In other words f should be a function such $f(x') = y'$ where (x', y') was also generated from the same probability distribution $P(x, y)$.

Accuracy of a function is measured by the zero-loss function $\frac{1}{2}|f(x) - y|$ where loss is 0 on correct classification and 1 on incorrect. We can now define two risk or error functions.

- Expectation of True Error:

$$R[f] = \frac{1}{2}\int |f(x) - y|\ dP(x, y) \tag{8}$$

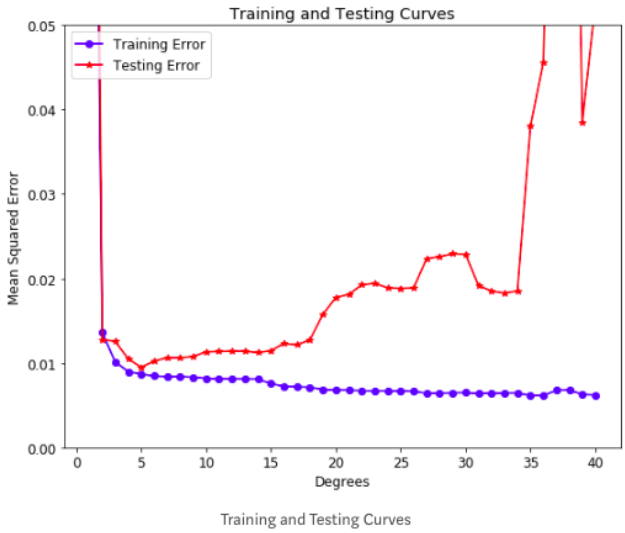

Figure 2: The graph shows how the Testing error increases with more complex models. Here the increase in Degrees imply the increase in degree of a fitted polynomial function f.

This error is the average error over sample points (test points) that were drawn from the underlying distribution P(x,y). This is in general also known as test-error.

- Average Empirical Risk:

$$R_{emp}[f] = \frac{1}{2m} \sum_{i=1}^{m} |f(x_i) - y| \tag{9}$$

This error is called the training-error as it is observed only on training data.

As, the underlying distribution is unknown, the main problem in learning theory is to find a function that performs well on the training and test data sets. A function, that minimizes the training data might not minimize the test data. According to Statistical LEarning Theory, a function genaralizes well if $|R[f] - R_{emp}[f]|$ is small. In other words the empirical risk converges to the expectation of true risk. [2]

This issue is commonly referred to as the bias-variance tradeoff. If a simple model is used to find the function f that generalizes well on unseen data then it will have high bias (selection-bias). It might also have high training error. However, if we choose a complicated model that works very well with the training samples, we might have high error on unseen data. This model will then have

high variance and low training error. Hence, the tradeoff is to choose a model complex enough to genaralize well on both training and test data points. In (Figure: 2) we can see how the training error and test error change with increasing complexity.

4.1 Vapnik–Chervonenkis Theory (VC)

The VC theory provides a way to restrict the set of functions from which f is chosen, to one which has a capacity(complexity, expressive power, richness, or flexibility) suitable for the amount of training data. There are many capacity measures like the VC dimension, annealed VC entropy or the fat shattering dimension. We will focus on the VC Dimension and hence the VC Bound.

Now, each function of the class, labels the training patterns in a certain way. Since the labels are in $\{\pm 1\}$, there are at most $2m$ different combination of labels for m patterns. However, a given class of functions might not be sufficiently rich to induce all these combinations; in other words, it might not be able to shatter the m points. A function with some parameters α, is said to **shatter** n points if for all possible assignments of labels to those points, there exists some α such that the model makes no errors when evaluating that set of data points. Similarly, for a set of functions, we can say that the set shatters n points when all the functions in the set can shatter at least n points.

VC Dimension is therefore defined, as the largest m such that there exists a set of m points which the class of functions can shatter, and ∞ if no such m exists.

Now, the **VC Bound** is defined as: Let $h < m$, be the VC dimension of the class of functions that the learning machine can implement, then for all functions of that class, with a probability of at least $1 - \sigma$ the following bound holds:

$$R[f] \leq R_{emp}[f] + \phi\Big(\frac{h}{m}, \frac{\log \sigma}{m}\Big) \tag{10}$$

here the confidence term (or capacity term) ϕ is defined as:

$$\phi\Big(\frac{h}{m}, \frac{\log \sigma}{m}\Big) = \sqrt{\frac{h(\log \frac{2m}{m} + 1) - \log(\sigma/4)}{m}} \tag{11}$$

Here also the bias-variance tradeoff can be seen in play. The capacity term ϕ is an increasing function of h. This means that with increase in h, the model will become more complex as it will be able to shatter more points. However, the confidence term increases and therefore the bound on the test error increases. Hence, the training error might drop but as the capacity term ϕ increases the bound does not necessarily lessen on the test-error.

In order to get nontrivial predictions from (10), the function set must be restricted such that its capacity (VC dimension) is small enough (in relation to the available amount of data). At the same time, the class should be large enough to provide functions that are able to model the dependencies hidden in $P(x, y)$. The choice of the set of functions is thus crucial for learning from data.

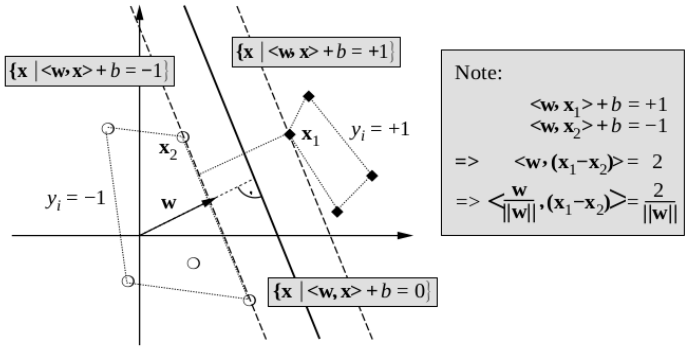

Figure 3: Optimal hyperplane

5 Support Vector Classification

Vapnik considered the class of hyperplanes in some dot product space H to be, $< \boldsymbol{w}, \boldsymbol{x} > +b = 0$, $\boldsymbol{w} \in H, b \in R$ corresponding to the decision function $f(\boldsymbol{x}) =< \boldsymbol{w}, \boldsymbol{x} > +b$. This learning algorithm's assumption is that the class labels are separable by the hyperplane, that is they are linearly separable. Among all the possible hyperplanes there exists a unique optimal hyperplane, distinguished by the maximum margin of separation between any training point and the hyperplane. The optimal hyperplane's form is:
$\max_{\boldsymbol{w},b}\ \min(||\boldsymbol{x} - \boldsymbol{x_i}|| : \boldsymbol{x} \in H, < \boldsymbol{w}, \boldsymbol{x} > +b = 0, i = 1, ..., m)$

Moreover, the capacity of the set of possible hyperplanes decreases with increase in margins. Hence, the VC bound improves. Therefore we can say that the optimal hyperplane has a good generalization performance. Additionally, it is computationally attractive, since it can be constructed by solving a quadratic programming problem for which there exist efficient algorithms.

The optimal hyperplane in (Figure: 3) is shown as a solid line. There exists a weight vector $\boldsymbol{w}$ and a threshold b such that $\mathrm{y}_i(< \boldsymbol{w}, \boldsymbol{x_i} > +b) > 0$ $(i = 1, ..., m)$. Re-scaling $\boldsymbol{w}$ and b such that the point(s) closest to the hyperplane satisfy $|<\boldsymbol{w}, \boldsymbol{x_i} > +b| = 1$, we obtain a canonical form $(\boldsymbol{w}, b)$ of the hyperplane, satisfying $\mathrm{y}_i(< \boldsymbol{w}, \boldsymbol{x_i} > +b) \geq 1$. In this case, the margin, measured perpendicularly to the hyperplane, equals $2/||\boldsymbol{w}||$. This can be seen by considering two points $\mathbf{x}_1, \mathbf{x}_2$ on opposite sides of the margin, that is, $<\boldsymbol{w}, \mathbf{x}_1 > +b = 1$, $< \boldsymbol{w}, \mathbf{x}_2 > +b = -1$ and projecting them onto the hyperplane normal vector $\boldsymbol{w}/||\boldsymbol{w}||$.

Now as we want to maximize the margin which is $2/||\boldsymbol{w}||$. We can create a minimizing function of $||\boldsymbol{w}||/2$. Therefore, to construct the optimal hyperplane, one has to solve the following constrained optimization problem [4, 3]:

$$\min_{\boldsymbol{w} \in H, b \in R} f(\boldsymbol{w}) = \frac{1}{2}||\boldsymbol{w}||^2 \tag{12}$$

$$\textit{subject to}\ \ \mathrm{y}_i(< \boldsymbol{w}, \boldsymbol{x_i} > +b) \geq 1$$

This problem can be solved using Lagrangian Multipliers.

5.1 Lagrangian and Strong Duality

Consider the general constrained optimization problem: $\min f(z)$ subject to $c_i(z) \geq 0$ for $i = 1, ..., m$ where f is the main minimization problem and the c_i are constraints on choosing z. This is called the primal problem. The corresponding Lagrangian is defined as:

$$L(z, \lambda) = f(z) - \sum_{i=1}^{M} \lambda_i c_i \tag{13}$$

with Lagrangian multipliers (also called dual variables) $\lambda_i \geq 0$. Based on the Lagrange function we can create a new function that provides a lower bound on the objective function f . Since the dual variables are all positive, i.e. $\lambda_i \geq 0$, we know that $f(z) \geq L(z, \lambda)$ for all feasible z. This motivates the definition of the Lagrange dual function as:

$$g(\lambda) = \inf_z L(z, \lambda) = \inf_z \left(f(z) - \sum_{i=1}^{M} \lambda_i c_i \right) \tag{14}$$

The dual function g is concave, even if the original problem is convex since it is the point-wise infimum of affine functions. The dual form yields lower bounds on the optimal value $p*$ of the objective function f . For any $\lambda \geq 0$ we have $g(\lambda) \leq p^*$. Therefore, the Lagrangian dual problem is to maximize g so that the lower bound on p^* is as close to it as possible. The Lagrangian dual problem is:

$$\max_{\lambda} g(\lambda) \quad s.t \quad \lambda_i \geq 0 \tag{15}$$

Now we can define strong duality using Slater's Conditions where the minimum of the primal problem coincides with the maximum of the dual problem, i.e. $d^* = \max_\lambda g(\lambda) = \inf_z f(z) = p^*$

5.2 Slater's Condition

Let's again consider the constrained optimization problem in feature space H: minimize $f(z)$ subject to $c_i(z) \leq 0$, $i = 1, 2, \ldots, m$.

Then the Slater's condition [5] states that:

- If $f, c_1, ..., c_m$ are convex and there exists an $z \in H$ such that $c_i(z) < 0$, for all $i = 1, 2, \ldots, m$
- Or f is convex and the constraints are affine, that is $c_i(z) = w_i^T z + b_i \leq 0$, for all $i = 1, 2, \ldots, m$

Then strong duality holds.

Now we can see that in the case of Support Vector Classification, the primal problem is convex and the constraints are all affine. Hence, strong duality holds. Therefore on maximizing the dual problem, we can find the minimum of the primal problem.

5.3 Karush-Kuhn-Tucker Conditions (KKT Conditions)

These conditions on Lagrangian multipliers along with strong duality are used to further solve. If we have the following optimization problem minimize $f(z)$ subject to $c_i(z) \geq 0, i = 1, ..., m$ with $f, c_1, ..., c_m$ convex and strong duality holds, then the following statement is true:

z^* and λ^* are the optimal solutions of the constrained optimization problem and the corresponding Lagrange dual problem if and only if they satisfy the Karush-Kuhn-Tucker (KKT) conditions:

$$c_i(z^*) \geq 0 \qquad \textit{primal feasibility}, \tag{16}$$

$$\lambda_i^* \geq 0 \qquad \textit{dual feasibility}, \tag{17}$$

$$\lambda_i^* c_i(z^*) = 0 \qquad \textit{complementary slackness}, \tag{18}$$

$$\nabla_z L(z^*, \lambda^*) = 0 \qquad z^* \textit{minimizes Lagrangian} \tag{19}$$

$$\text{for all } i = 1, ..., m$$

5.4 Solving Support Vector Classification Algorithm

The Lagrangian of this model will be given by combining equations 12 and 13:

$$\mathrm{L}(\boldsymbol{w}, b, \boldsymbol{\lambda}) = \tfrac{1}{2}||\boldsymbol{w}||^2 - \textstyle\sum_{i=1}^m \lambda_i (y_i(< \boldsymbol{x_i}, \boldsymbol{w} > + b) - 1)$$

here $\boldsymbol{\lambda} = (\lambda_1, ..., \lambda_m)$. Now the derivatives of L w.r.t $\boldsymbol{w}$ and b is: $\nabla_w L = w + \sum_{i=1}^m \lambda_i y_i \boldsymbol{x_i}$ and $\nabla_b L = \sum_{i=1}^m \lambda_i y_i$ As strong duality holds and the primal and constraints are all convex functions, we can use KKT conditions to further solve this problem. Now we can equate the derivative of Lagrangian to 0 (19). This gives us:

$$\boldsymbol{w} = \sum_{i=1}^m \lambda_i y_i \boldsymbol{x_i} \quad \textstyle\sum_{i=1}^m \lambda_i y_i = 0 \tag{20}$$

This further leads to the dual problem of

$$\begin{aligned} \max_{\boldsymbol{\lambda}} g(\boldsymbol{\lambda}) &= \frac{1}{2} \sum_{i,j=1}^m \lambda_{ij} y_i y_j \boldsymbol{x_i} \boldsymbol{x_j} - \sum_{i,j=1}^m \lambda_{ij} y_i y_j \boldsymbol{x_i} \boldsymbol{x_j} - b \sum_{i=1}^m \lambda_i y_i + \sum_{i=1}^m \lambda_i \\ &= \sum_{i=1}^m \lambda_i - \frac{1}{2} \sum_{i,j=1}^m \lambda_{ij} y_i y_j \boldsymbol{x_i} \boldsymbol{x_j} \end{aligned} \tag{21}$$

subject to $\lambda_i \geq 0,\ i = 1, ..., m$ and $\sum_{i=1}^m \lambda_i y_i = 0$. This is solved using a Quadratic program (QP) solver for $\boldsymbol{\lambda}$ through which $\boldsymbol{w}$ and b can also be solved. The final discussion is of the form:

$$f(\boldsymbol{x}) = sgn\Big(\sum_{i=1}^m y_i \lambda_i^* \boldsymbol{x} \boldsymbol{x_i} + b^* \Big) \tag{22}$$

Here b^* and $\lambda_i^* s$ are the optimal values of b and $\lambda_i s$.

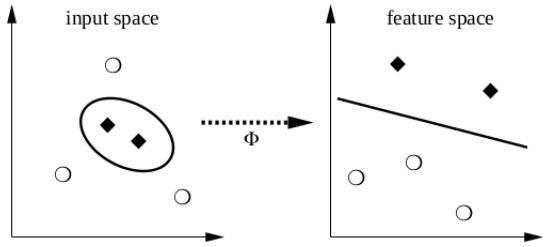

Figure 4: Kernel on Input Space

Combining equation (20) and the complementary slackness condition under KKT:
For λ_i non-zero we have $y_i(< \boldsymbol{x_i}, \boldsymbol{w} > +b) - 1 = 0$. This implies that for λ_i non-zero the $\boldsymbol{x_i}$ lies on one of the margins. These are the points that contribute to the value of $\boldsymbol{w}$ and play a role in the optimization. As for all other $\lambda_i = 0$, $\boldsymbol{w}$ does not change. These data points are called **support vectors**.

5.5 Kernel Trick

If the input space is not linearly separable (Figure 4) then a mapping ϕ can be used to map the training data into a higher-dimensional feature space where the points are linearly separable. This yields a nonlinear decision boundary in input space. By the use of a kernel function, it is possible to compute the separating hyperplane without explicitly carrying out the mapping into the high-dimensional feature space. This is referred to as the kernel trick. Now we have a dot product in our earlier decision function (22) $< \boldsymbol{x}, \boldsymbol{x_i} >$ which can be written as:

$$< \boldsymbol{x}, \boldsymbol{x_i} >=< \phi(x), \phi(x_i) >= k(x, x_i) \tag{23}$$

The kernel trick can be applied since all feature vectors only occurred in dot products. The weight vector then becomes an expansion in feature space, and therefore will typically no longer correspond to the ϕ-image of a single vector from input space. We thus obtain decision functions of the form:

$$f(\boldsymbol{x}) = sgn\Big(\sum_{i=1}^{m} y_i \lambda_i^* k(x, x_i) + b^*\Big) \tag{24}$$

which is obtained through solving the quadractic problem:

$$\max_{\boldsymbol{\lambda}} g(\boldsymbol{\lambda}) = \sum_{i=1}^{m} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{m} \lambda_{ij} y_i y_j k(x_i, x_j) \tag{25}$$

subject to $\lambda_i \geq 0,\ i = 1, ..., m$ and $\sum_{i=1}^{m} \lambda_i y_i = 0$.

6 Kernel PCA

Standard PCA reduces the dimensionality of the observed data by projecting it onto a linear subspace. The projection is chosen in a way that the error measured in the squared standard Euclidean norm is minimized.

Let $\boldsymbol{X} \in \boldsymbol{R}^{p*n}$ be the centered data matrix and let $\boldsymbol{K} := \boldsymbol{X^T X}$ be the (n n)-matrix consisting of all inner products of the data. This is called the Gram Matrix. More precisely, the (i, j) entry of $\boldsymbol{K}$ is the inner product $\boldsymbol{x_i^T x_j}$ of the i-th and the j-th input points.

Let $\boldsymbol{X} = \boldsymbol{U\Sigma V^T}$ be the SVD of the data matrix, then the first k principle components of a data vector $\boldsymbol{y} \in \boldsymbol{R}^p$ are given by $\boldsymbol{U_k^T y}$. Here the eigen-value decomposition of $\boldsymbol{K}$ helps. Replacing the value of $\boldsymbol{X}$ in $\boldsymbol{K}$, we get:

$$\boldsymbol{K} = \boldsymbol{V\Sigma^T\Sigma V^T} \tag{26}$$

Here, $\boldsymbol{V}$ is orthonormal and $\boldsymbol{\Sigma^T\Sigma}$ is the diagonal. Thus, Equation (26) is the eigenvalue decomposition of $\boldsymbol{K}$, due to the uniqueness of the Eigen Value Decomposition, and by computing the k largest eigenvalues of $\boldsymbol{K}$ with their respective eigenvectors, we obtain $\sigma_1^2, ..., \sigma_k^2$ and $\boldsymbol{V_k}$. We can now also write $\boldsymbol{\Sigma_k}$ as $diag(\sigma_1, ..., \sigma_k)$.

Now to compute $\boldsymbol{U_k^T y}$ we first have:

$$\begin{aligned} \boldsymbol{V_k^T X^T y} &= \boldsymbol{V_k^T V\Sigma U^T y} \\ &= [I_k|0]\boldsymbol{\Sigma U^T y} \\ &= [\boldsymbol{\Sigma_k}|0]\boldsymbol{U^T y} \\ &= \boldsymbol{\Sigma_k U^T y} \end{aligned}$$

By multiplying this equation with $\boldsymbol{\Sigma_k^1}$, we obtain:

$$\begin{aligned} \boldsymbol{U^T y} &= \boldsymbol{\Sigma_k^1 V_k^T X^T y} \\ &= \boldsymbol{\Sigma_k^1 V_k^T}[\boldsymbol{x_1^T y}, ..., \boldsymbol{x_n^T y}]^{\boldsymbol{T}} \end{aligned} \tag{27}$$

Now we can see that for a new measurement $\boldsymbol{y}$, the first k principal components is given by eq (27) which can now be computed only through inner products of the data, namely the Gram Matrix $\boldsymbol{K}$ and the inner products of $\boldsymbol{x_i^T y}$. This can also be extended for a general kernel function by replacing the inner product $\boldsymbol{x^T y}$ by $< \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) >= k(\boldsymbol{x}, \boldsymbol{y})$

7 Common Kernel Functions

Some common kernel functions used are:

- Polynomial Kernel: k(x, x') = <x, x'>^{d}, where d denotes the degree.
- Gaussian or Radial Basis Kernel: k(x, x') = $\exp\left(\frac{-||x-x'||2}{2\sigma^2}\right)$ (In Python the formula used for RBF kernel is $\exp(-\sigma||x - x'||^2)$

- Sigmoid Kernel: $k(x, x') = \tanh(\gamma < x, x' > +\phi)$

By the use of kernels, the optimal margin classifier was turned into a high-performance classifier. Surprisingly, it was noticed that the polynomial kernel, gaussian and sigmoid kernel with suitable choices of $d \in \boldsymbol{N}$ and $\sigma, \gamma, \phi \in \boldsymbol{R}$ empirically led to SV classifiers with very similar accuracies. In this sense, the SV set seems to characterize (or compress) the given task in a manner which to some extent is independent of the type of kernel (that is, the type of classifier) used.

8 Results from Python Implementation

Parameters	Polynomial degree = 1	RBF gamma = 19.6	Sigmoid gamma = 1	Classes
Accuracy	0.93	0.96	0.96	
Precision	0.92	1	0.93	1
	0.94	0.94	1	2
Recall	0.92	0.92	1	1
	0.94	1	0.94	2
F1-Score	0.92	0.96	0.96	1
	0.94	0.97	0.97	2

Figure 5: Results from Support Vector Classification for 2 classes

Parameters	Polynomial degree = 3	RBF gamma = 1.20	Sigmoid gamma = 0.3	Classes
Accuracy	0.97	0.97	0.91	
Precision	1	1	1	0
	0.94	1	0.82	1
	1	0.94	0.93	2
Recall	1	1	1	0
	1	0.93	0.93	1
	0.94	1	0.81	2
F1-Score	1	1	1	0
	0.97	0.97	0.87	1
	0.97	0.97	0.87	2

Figure 6: Results from Support Vector Classification for 3 classes

From the summaries in figures 5 and 6, it can be clearly seen that with the right choice of the parameters, once can get a same level of performance from different kernels. Hence, Support Vector Classification algorithm in itself is a very strong tool.
Decision boundaries results for 2 class and 3 class SV classifier with best performing kernels:

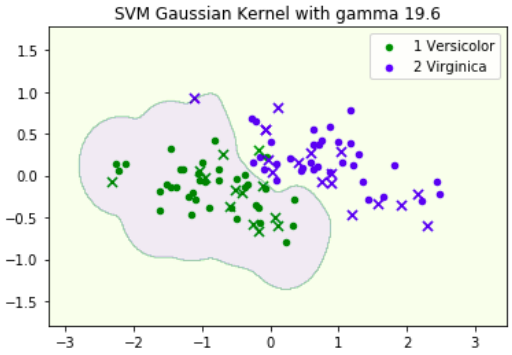

Figure 7: Decision Boundary for RBF kernel for 2 classes

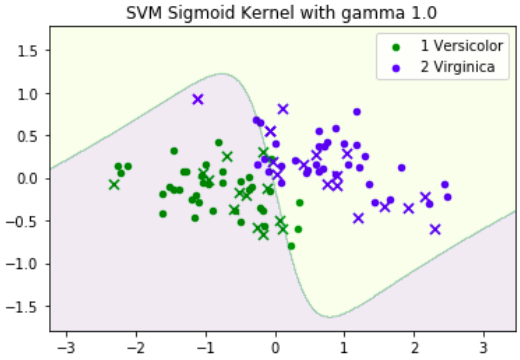

Figure 8: Decision Boundary for Sigmoid kernel for 2 classes

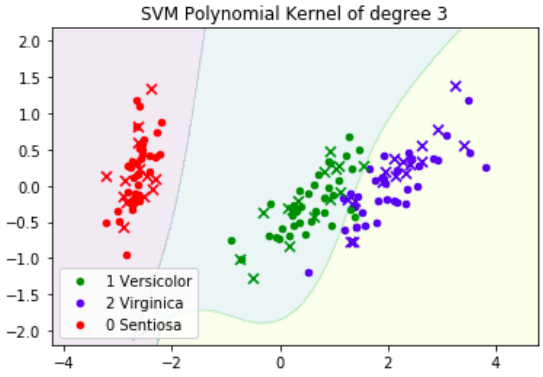

Figure 9: Decision Boundary for Polynomial kernel for 3 classes

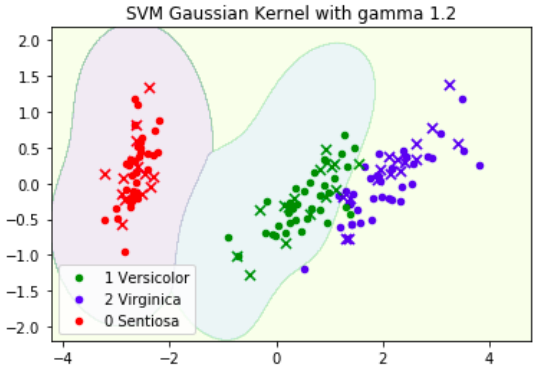

Figure 10: Decision Boundary for RBF kernel for 3 classes

9 Summary

SV classifiers have proved to be a strong classification algorithm because of easy computations and mappings to feature space through kernels. Moreover, many Quadractic Solvers are readily available to solve the dual problem. The RBF kernel in general performs well, as we saw from Section 8. SV classifier has fewer parameters to train or tune. Hence, they are faster. Most importantly, it is a convex problem and therefore guarantees a global optimum.

References

[1] Bernhard Schölkopf and Alexander J. Smola, *"Learning with Kernels"* In `https://www.cs.utah.edu/~piyush/teaching/learning-with-kernels.pdf`, pages 1–22, Massachusetts Institute of Technology, 2000.

[2] Ulrike von Luxburg and Bernhard Schölkopf, *"Statistical Learning Theory: Models, Concepts, and Results"* In `http://www.tml.cs.uni-tuebingen.de/team/luxburg/publications/StatisticalLearningTheory.pdf`, 2008.

[3] Christopher M. Bishop, *"Pattern Recognition and Machine Learning"* "Springer", pages 291–353, 2006.

[4] Kevin P. Murphy, *"Machine Learning: A Probabilistic Perspective"* "The MIT Press", pages 475–512, 2012

[5] Stephen Boyd and Lieven Vandenberghe, *"Convex Optimization"* In `https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf`, pages 215 – 250, Cambridge University Press, 2004.