American University *of* Armenia

Հայաստանի Ամերիկյան Համալսարան

Support Vector Machines

May 20, 2018

Instructor

Michael Poghosyan

Members

Henrik Sergoyan

Hilda Karakhanyan

Seda Sedrakyan

Abstract

Support Vector Machines have a big optimization role in everyday life and not only. They use special techniques for classification and regression of data for optimal use. There are special support vector machine algorithms and methods with help of which data is being classified in relatively right, optimized and useful way. Thus the paper discusses two methods to understand Support Vector Machines problem using Maximum Margin classifier and Kernel method.

Keywords: Support Vector Machines, Data classification, Maximum Margin, Kernel

Content

1.Introduction

1.1Problem setting and description

In everyday life, people use lots of applications and devices which work is based on Support Vector Machine. Support Vector Machines (SVM) are supervised learning models with learning algorithms for analyzing data for classification and regression analysis. There are special methods of support vector machine for getting classified data. The paper gives information about SVM in possible understandable and useful way. The methods which are discussed in this paper and are actually widely used in the support vector machine theory are Maximum Margin classifier and Kernel method.

1.2Structure of the paper

The paper consists of five main parts. The first part gives an introduction to the paper and structure. In the second part, the paper discusses topics to better understand the theory of support vector machine. Mainly it gives answers about what is SVM, how and where are they being used, when did the theory appear. Then background information part gives some understanding of hyperplanes and Kernel function. In the third part, the paper discusses topics related to Statistical learning such as binary classification, Loss function. Further, it goes deeper in Maximum Margin Classifiers. In the fourth part, the paper comes to culmination discussing Kernel method which gives the optimal solution to many, classification, regression, and other similar problems. The fifth part gives a conclusion to the paper and in the end, the paper has some useful code interpretation of what we have done so far.

2.Theoretical Background

2.1 What is SVM?

Support Vector Machine (SVM) is supervised learning models with learning algorithms which analyze data for classification and regression analysis. However, most often SVM is used in classification problems. For instance, when there is a set of training examples divided into two categories and the function of SVM training algorithm is building the model to understand in which group the new example needs to be included. SVM can perform both linear and non-linear classification. Non-linear classification is being done by kernel trick, which maps inputs into nD feature spaces and this method is explained further in the paper.

2.2 SVM Applications and Usages

The main purpose of using Support Vector machines is to correctly classify still unseen data. From the first sight, this can seem hard understandable however Support Vector machines have many applications in various fields. In our everyday life, we also meet the use of Support vector machines such as face detection and image classification, text and hypertext categorization additionally detecting handwritten texts and signatures by different electronics. Support Vector Machines are also used in science such as bioinformatics. To better understand all of them lets just look at each of above-mentioned examples one by one.

Face detection: If one ever had a smartphone or photographing camera he or she should notice that while photographing someone a square appears on that particular person's face on the screen. Here SVMs help to classify all the parts of the image into two groups one of which is face denoted by +1 and another one is non-face denoted by -1. Here parts are the pixels of the picture and based on their brightness SVM creates a square boundary around the face and each image is classified by the same process. Support vector machines are considered to be very accurate classifiers for images with high search accuracy.

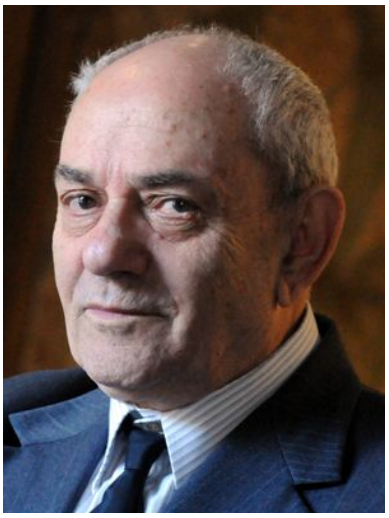
Text and hypertext categorization: allows text and hypertext categorization using training data to classify documents into different categories such as scientific papers, emails, books and soon. In addition to this support vector machines can also classify emails into "business" and "personal" and many other subgroups based on need and variables used in the algorithm. Here classification is based mainly on the threshold rate that particular document or file has. As a main and important step in such kind of problems first, the score of the document is being calculated and compared with a predefined threshold and based on the comparison the document is being classified into a definite category. When the score of the text is not passing the threshold it is being classified as a general document. IN addition to these SVMs help also to detect handwritten texts and validate the signatures on them.

Bioinformatics: One of the most common problems of bioinformatics is the protein remote homology detection. The support vector machines are considered to be the most effective method of solving this problem. Here the big role is in models of protein sequences. SVM algorithms are used also in the classification of genes.

Concluding this topic it is important to mention that the SVM helps people by making reliable predictions, reducing redundant information and in many other important fields when one even cannot imagine.

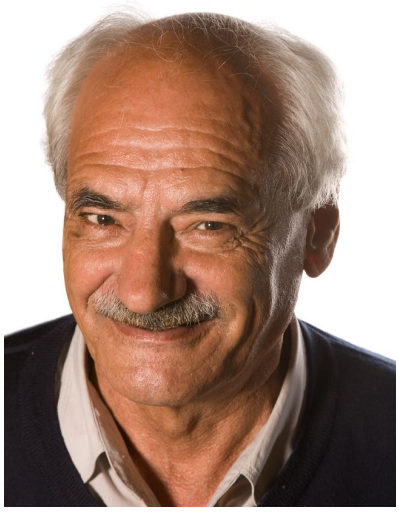
2.3 History of SVM

The original SVM algorithm was invented in 1963. Inventors were Vladimir Vapnik and Alexey Chervonenkis. Later the development of science brought to more advanced needs and in 1992, Bernhard Boser, Isabelle Guyon, and Vladimir Vapnik started the process of creating nonlinear classifiers. That was by applying the kernel trick to maximum-margin hyperplanes. The final version of them has formed already in 1993 and published in 1995. Now SVMs become more and more popular and used in different aspects some of which were mentioned above.



Vladimir Vapnik

Vladimir Vapnik was born on December 6 in 1936. He was one of the developers of Vapnik-Chervonenkis statistical learning theory. He also was among the members who invented Support Vector Machine method and latter support vector clustering algorithm .



Alexey Chervonenkis

Alexey Chervonenkis was born on September 7 in 1938. With Vladimir Vapnik, as mentioned above, they developed Vapnik-Chervonenkis theory which is known by the name “Fundamental Theory of Learning”. A theory is very important part of the computational theory. On September 22 of 2014 got lost and later was found dead because of hypothermia.

Bernhard E. Boser

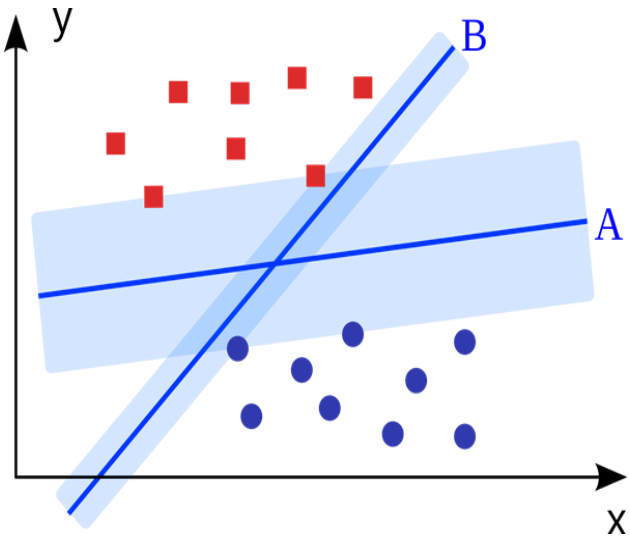
Professor Boser received B.S. degree from the Swiss Federal Institute of Technology in 1984. He received M.S. and Ph.D. degrees from Stanford University in 1985 and 1988 respectively. Currently, he is a professor of Electrical Engineering and Computer Science at the University of California, Berkeley, a Co-Director of the Berkeley Sensor & Actuator Center, a Co-Director of the UC Berkeley Swarm Lab.

Isabelle Guyon

Dr. Guyon is co-inventor of the support vector machine (SVM) algorithm, and SVM-RFE feature selection, two machine learning methods essential for scientific prediction in biomedical informatics. Dr. Guyon is considered to be one of the most cited researchers in machine learning in all over the world. She has developed methods for biomedical informatics and has applied them in ways that can significantly develop the area.

2.4 Definitions and concepts

Support vector machine constructs a hyperplane or set of hyperplanes for classification, regression, outliers detection etc. Support vector machine in case of need also constructs infinite-dimensional space hyperplane for some particular problems. Thus, for instance, if the task is a classification of data SVM does it by creating hyperplane. Consequently, there can be some hyperplanes which satisfy the problem, however, the best one will be the one which has the largest distance to the nearest training data point of any class. Actually, this is for reducing the possibility of error of the classifier. Thus, discussing the situation of the



graph as the largest distance of nearest training data has A consequently in this situation A will be considered as the best classifier hyperplane.

In addition to simple cases of linearly separable sets, there are also sets which cannot be separated linearly. For that case, the solution is that finite-dimensional space needs to be mapped into a higher-dimensional space. However, in this case, computation becomes complex enough as generates need of computing dot products of high-dimensional vectors. For the solution of this problem was developed Kernel method which suggests computations of variables in the original space by defining them in terms of Kernel function K(x,y).

Now the paper will give deeper information about the statistical learning and will illustrate deeper theoretical understanding of SVM algorithms.

3. Algorithms for SVM and optimization solutions

3.1 Statistical Learning

Statistical learning deals with the problem of finding a functional relationship based on finite data, which in its turn helps to predict the output for future inputs. It has its wide applications in bioinformatics, facial recognition, pattern recognition, and in other spheres where forecasting methods are needed. Statistical learning is often categorized into two main categories - supervised learning and unsupervised learning. Supervised Learning in its turn deals with

problems of classification and regression. Diagnosing a patient with diabetes using clinical parameters data is a widely used classification problem which is also solved using Support Vector Machine algorithms.

3.2 Binary Classification problem

Binary classification is defined as categorizing the elements of a dataset into two groups, based on some classification rule. It can be applied when we need, for example, to decide whether the patient will have the disease or not, or whether the student will pass the exam or fail based on the training data. So, the goal of binary classification is to compute an output that only has two states. Thus, we define the set of possible output values by Y = {−1, +1}.

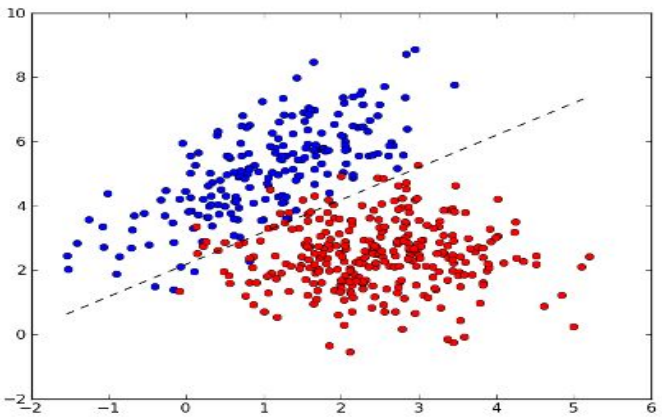


Figure 3.2.1

3.3 Loss Function

Loss function helps to quantify the loss or cost associated with the errors committed while estimating a parameter. The goal of optimization is to minimize the loss function. Loss function can be defined in many ways, for example,

$$L(y, f(x)) = |y - f(x)| \quad \text{or} \quad L(y, f(x)) = (y - f(x))^2$$

where, f(x) is our estimation for y. We assume that our quality measure is a non-negative real number L(y, f(x)) that is smaller when the estimated response f(x) is better. However, in statistical learning we are interested in the expected value of the loss function, which can be defined as,

$$R_{L,P}(f) = E(L(y, f(x)))$$

Where, f has the probability distribution P. So our problem becomes to minimize $R_{L,P}(f)$, usually we can't calculated this expected value, because we do not know the distribution of (X, Y). By the *Law of Large Numbers*,

$$R_{L,P}(f) \approx \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i))$$

So, instead we minimize the quantity above.

Throughout this paper we are going to consider problems of binary classification using SVM, so for our classification problem the loss function L_{class} equals 1 if $sign(f(x)) = y$ and equals 0 otherwise. The empirical risk for the L_{class} will be

$$R_{L_{class},\,D}(f) \;\approx\; \frac{1}{n}\sum_{i=1}^{n} L_{class}(y_i,\, f(x_i))$$

where $D = ((x_1, y_1), \ldots, (x_n, y_n))$ is the finite sequence of already gathered data. Unfortunately, however, minimizing $R_{L_{class},\,D}(f)$ over all functions doesn't lead to a good approximation and can cause *overfitting*. One way to avoid overfitting is to choose a small set F of functions f : X → R and minimizes only over F - This is known as Empirical Risk Minimization. The loss function for binary classification described above is non-convex which makes the computation of $R_{L_{class},\,D}(f)$ hard and even impossible. To make the minimization problem of $R_{L_{class},\,D}(f)$ computationally possible SVM replaces the loss function by convex surrogate function called Hinge Loss function and is defined as

$$L_{hinge(y,\,t)} = max\{0,\; 1 - yt\} \text{ where } y \in \{-1,\; 1\},\; t \in R$$

Then $R_{L_{hinge,}D}(f)$ is convex and assuming that F is a convex set, we only are left with convex optimization problem. Regarding to computational feasibility we will consider very specific sets of functions - reproducing kernel Hilbert spaces in one of the upcoming sections.

3.4 Maximum Margin Classifier

3.4.1 Hyperplane

In a n - dimensional space, A hyperplane is a subspace whose dimension is n - 1. If n = 3 then hyperplanes will be planes, if n = 2, its hyperplanes are lines. This notion can be used in any general space in which the concept of the dimension of a subspace is defined. So, in 2D space a hyperplane can be defined by the line equation

$$\beta + w_1 X_1 + w_2 X_2 = 0$$

And in the same way for the n-D space hyperplane will be defined as

$$\beta + w_1 X_1 + w_2 X_2 + \ldots + w_n X_n = 0 \quad \text{or} \quad \beta + w^T X = 0 \quad (1),$$

in vector notation, where $X = (X_1, X_2, \ldots, X_n)^T$. If X does not satisfy the equation of hyperplane above then it is either

$$\beta + w^T X > 0 \quad (2) \quad \text{or} \quad \beta + w^T X < 0 \quad (3)$$

So, hyperplane divides the n-D space into three parts (1), (2), (3).

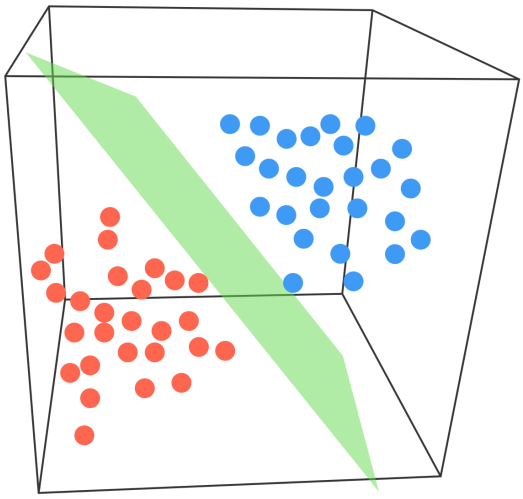


Figure 3.4.1.1: example of a 2D hyperplane-plane in 3D space.

3.4.2 Maximizing margins

Suppose we have a $D = ((X_1, Y_1), \ldots, (X_N, Y_N))$ dataset, where $X_k \, \varepsilon \, R^n$ are our feature parameters, and $X_k = (x^k{}_1, \ldots, x^k{}_n)$ and $Y_k \, \varepsilon \, (-1, +1)$, $k = 1, \ldots, N$. We will say

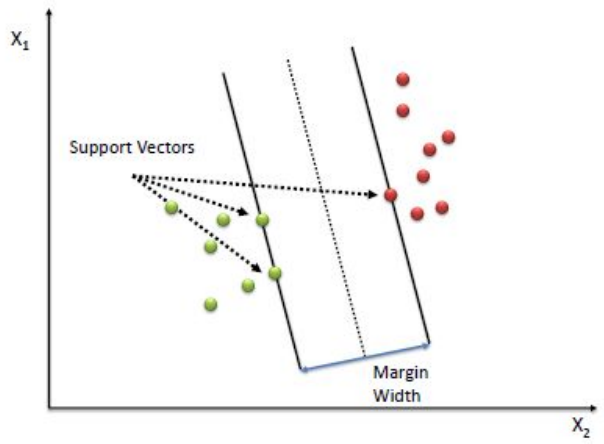
$$\beta + w^T X > 0 \quad \text{when} \quad y = +1$$

$$\beta + w^T X < 0 \quad \text{when} \quad y = -1$$

The idea of maximum margin hyperplane will be to generate the maximum margin and, and for that margin choose the middle hyperplane. we can compute the perpendicular distance from each training data observation to a given separating hyperplane; the smallest such distance is the minimal distance from the observations to the hyperplane, and is known as the margin.

3.4.3 The Separable Case: Hard Margin

Figure 3.4.3.1



By convention, among many representations of the hyperplane, the one chosen is

$$\left|w^T X + \beta\right| = 1 \qquad (4)$$

where x symbolizes the training data points which are closest to the hyperplane. These observations are known as the support vectors (see the figure 3.4.3). (4) defines two lines parallel to each other and the hyperplane in the middle, known as the Widest Street Approach. Opening (4) we get (5), (6)

$$w^T X + \beta = 1 \qquad (5)$$

$$w^T X + \beta = -1 \qquad (6)$$

So, now, we will say

$$\beta + w^T X \geq 1 \quad \text{when} \quad y = +1$$

$$\beta + w^T X \leq 1 \quad \text{when} \quad y = +1$$

To find the margin, let's subtract equation (6) from (5), we get

$$w^T (X_1 - X_2) = 2 \; 2$$

$$(X_1 - X_2)^T w = ||X_1 - X_2||\, ||w||\, cos(X_1 - X_2,\; w)$$

as w is normal vector for $w^T X + \beta = 0$ then

$$||X_1 - X_2||\, cos(X_1 - X_2,\; w) = ||X_2 - X_3||,$$

which is our margin (*see the figure 3.4.3.2*), and it equals

$$\| X_2 - X_3 \| = \frac{(X_1 - X_2)^T w}{\|w\|} = \frac{2}{\|w\|}$$

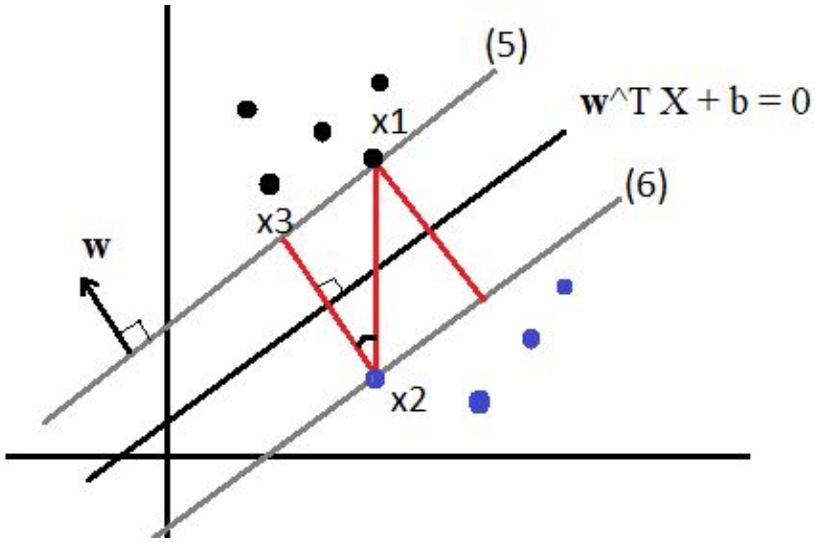


Figure 3.4.3.2

So, now our problem is to maximize the margin $\frac{2}{\|w\|}$, which implies we can minimize $\frac{\|w\|}{2}$, or equivalently minimize $\frac{\|w\|^2}{2}$, which is mathematically more convenient. Thus we get the following optimization problem

$$minimize \ \frac{\|w\|^2}{2}$$

$$subject\ to\ y_i(w^T X_i + b) \geq 1$$

The following optimization problem is usually solved by Lagrange function and Lagrange multipliers. The Lagrange function corresponding to our problem will be

$$L = \frac{||w||^2}{2} - \sum_{i=1}^{n} \lambda_i (y_i (w^T X_i + b - 1))$$

Sometimes, yet another constraint is added to the optimization problem above, which is $||w||^2 = 1$, and the Lagrange function changes correspondingly.

3.4.4 The Non-separable Case: Soft Margin

In many cases no separating hyperplane exists, and so there is no maximal margin classifier. Thus, for this case, the optimization problem described above doesn't have solution.

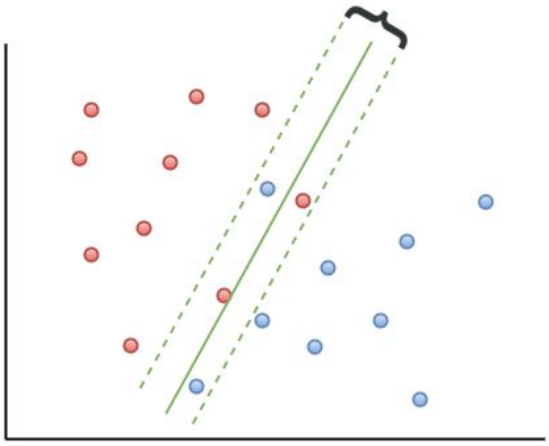


Figure 3.4.4.1

Here, in the Figure 3.4.4.1 it can be seen that we cannot exactly separate the two classes. However, as we will see, we can extend the concept of a separating hyperplane in order to develop a hyperplane that almost separates the classes, and the soft margin classifier does exactly this. Instead of finding the maximum margin so that every observation is not only on the correct side of the hyperplane but

also on the correct side of the margin, we instead allow some observations to be on the incorrect side of the margin, or even the incorrect side of the hyperplane. It is called soft, because here we soften the constraint to

$$y_i(w^T X_i + b) \geq 1 - \varepsilon_i , \quad \varepsilon_i \geq 0$$

So, let's restate our optimization problem for this non-linearly separable case.

$$\textit{minimize} \quad \frac{\|w\|^2}{2} + C \sum_{i=1}^{n} \varepsilon_i$$

$$\textit{subject to} \quad y_i(w^T X_i + b) \geq 1 - \varepsilon_i ,$$

$$\varepsilon \geq 0$$

Note: $\varepsilon = [\varepsilon_1, \varepsilon_2 ..., \varepsilon_n]$ is the vector of slack variables. If $\varepsilon_i = 0$ then the i-th observation is on the correct side of the margin. If $\varepsilon_i > 0$ then the i-th observation is on the wrong side of the margin, and we say that the i-th observation has violated the margin. If $\varepsilon_i > 1$ then it is on the wrong side of the hyperplane.

Thus the soft-margin SVM minimizes a convex objective function which is a Quadratic Program with 2N linear inequality constraints. C is a nonnegative tuning parameter which controls the trade-off between large margin vs small training error. C bounds the sum of the ε_i 's, and so it determines the number and severity of the violations to the margin (and to the hyperplane) that can be tolerated. C is often calculated by *cross-validation*.Small C allows constraints to be easily ignored which leads to large margin. Large C makes constraints hard to ignore which leads to

narrow margin. As C tends to infinity, the type of SVM is changing from soft to hard margin classifier. The optimization problem above is referred as Primal and changing it into Dual we get

$$maximize \qquad \sum_{i=1}^{n} \lambda_i - \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \lambda_i \lambda_j x_i^T x_j$$

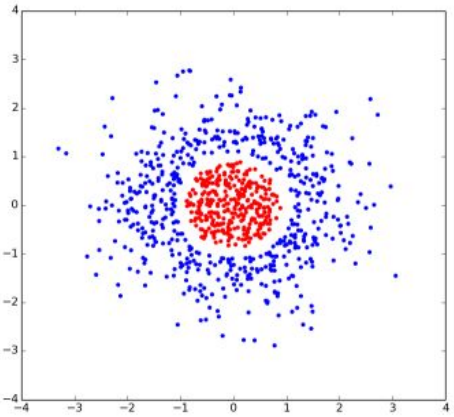
$$subject\ to \quad 0 \leq \lambda_i \leq C\,, \ \sum_{i=1}^{n} \lambda_i y_i = 0$$

$$i = 1, \ldots, n$$

Then in accordance to the problem above, the Lagrange function is formed and the problem is solved using Karush Kuhn Tucker (KKT) conditions.

4. Kernels

4.1 Kernel Optimization



What if our data is not linearly separable? Then we should enlarge our feature space and in addition to linear features we should use other nonlinear features. Nevertheless, choosing the best feature space could be very difficult task, and therefore,

idea of kernels was given, which shows similarity between two points.

Characterizing the similarity of the outputs $\{\pm 1\}$ is easy: in binary classification, only two situations can occur: two labels can either be identical or different. The choice of the similarity measure for the inputs, on the other hand, is a deep question that lies at the core of the field of machine learning.

Let us consider a similarity measure of the form

$$K : X \times X \to R;\ (x, y) \to K(x, y).$$

K is a function that, given two patterns x and y, which returns a real number characterizing their similarity. Unless stated otherwise, we will assume that K is symmetric, that is,

$$K(x, y) = K(y, x) \text{ for all } x, y \in X.$$

The function K is called a kernel.

In order to be able to use a dot product as a similarity measure, we therefore first need to represent x and y as vectors in some feature space H (Hilbert space). To this end, we use a map

$$\Phi: X \to H \qquad x \to \mathbf{x} := \Phi(x).$$

The freedom to choose the mapping Φ will enable us to design a large variety of similarity measures and learning algorithms.

In many situations, we cannot separate the data with a hyperplane. Instead, a nonlinear classification function is being designed which is more preferable for classification in such cases than linear classification function. Define function

$$H(x) = w^T \phi(x) + b,$$

where any point x giving $H(x) > 0$ will be recognized as +1, and any point x giving $H(x) < 0$ will be recognized as -1.

The only change appears from the previous optimization problems is that instead of using dot product of $w^T * x$ we use $w^T * \varphi(x)$.

$$\min \quad \tfrac{1}{2}||w||^2$$

$$\text{s.t.} \quad y_i [w^T \varphi(x) + b] \geq 1, \; i = 1, \ldots, n$$

Then generalized Lagrangian function is being formed as

$$L(w, b, \lambda) = \tfrac{1}{2}||w||^2 - \sum_{i=1}^{n} \lambda_i [\, y_i (w^T \varphi(x_i) + b) - 1]$$

Letting its partial derivatives with respect to w and b be zero,

$$\frac{\partial L(w, b, \lambda)}{\partial w} = 0 \rightarrow w = \sum_{i=1}^{n} \lambda_i y_i \varphi(x_i)$$

$$\frac{\partial L(w, b, \lambda)}{\partial b} = 0 \rightarrow \sum_{i=1}^{n} \lambda_i y_i = 0$$

Then eliminating the primal decision variables w and b, and get the objective of the Lagrange dual problem as

$$L(w,b,\lambda) = \tfrac{1}{2} w^T [\sum_{i=1}^{n} \lambda_i y_i \varphi(x_i)] - \sum_{i=1}^{n} \lambda_i y_i w^T \varphi(x_i) - \sum_{i=1}^{n} \lambda_i y_i b + \sum_{i=1}^{n} \lambda_i = -\tfrac{1}{2} \sum_{i=1}^{n} [\lambda_i y_i \varphi(x_i)]^T$$

$$\sum_{i=1}^{n} \lambda_i y_i \varphi(x_i) - b(\sum_{i=1}^{n} \lambda_i y_i) + \sum_{i=1}^{n} \lambda_i = \sum_{i=1}^{n} \lambda_i - \tfrac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \lambda_i \lambda_j [\varphi(x_i)]^T \varphi(x_i)$$

The whole dual problem can then be written as

$$\max \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \lambda_i \lambda_j [\varphi(x_i)]^T \varphi(x_i)$$

$$\text{s.t. } \lambda_i \geq 0, \; i = 1, \ldots, n$$

$$\sum_{i=1}^{n} \lambda_i y_i = 0$$

This problem needs to be solved with respect to λ, and sometimes it is easier to solve, because we do not need to know the detailed form of $\varphi(x)$. Instead, one only needs to know the Kernel function

$$K(x,y) = [\varphi(x)]^T \varphi(x)$$

The dual problem can then be written as

$$\max \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \lambda_i \lambda_j K(x,y)$$

$$\text{s.t. } \lambda_i \geq 0, \; i = 1, \ldots, n$$

$$\sum_{i=1}^{n} \lambda_i y_i = 0$$

This transformation usually is called kernel trick. The parameter w^* is then recovered from the solution λ^* of the dual optimization problem

$$w^* = \sum_{i=1}^{n} \lambda^*_i y_i \varphi(x_i)$$

Moreover, for the few $\lambda_i > 0$, the corresponding x_i satisfies

$$y_i [(w^*)^T \varphi(x) + b^*] = 1.$$

This means

$$(w^*)^T \varphi(x) + b^* = 1/y_i = y_i \quad \rightarrow \quad b^* = y_i - (w^*)^T \varphi(x)$$

In practice, it is more robust to average over all support vectors and calculate b^* as

$$b^* = \frac{1}{|S|} \sum_{i \in S} [y_i - (w^*)^T \varphi(x)]$$

Where S denotes the set of the indices of all support vectors and |S| is the cardinality of S.

For any new sample Z, the decision of classification can be given as

$$sign\left[(w^*)^T \varphi(x) + b^*\right] = sign\left[\sum_{i=1}^{n} \lambda_i^* y_i K(x_i, Z) + b^*\right]$$

Which can be determined without knowing the detailed form of φ(x).

Applying kernel functions provides a powerful tool to model possible nonlinear relations within data.

4.2 Examples of kernel functions and its interpretations

4.2.1 Linear kernel

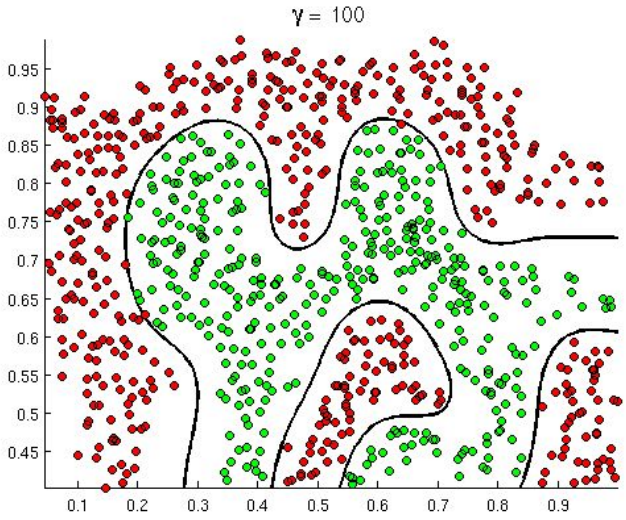
The simplest example of kernel function is linear kernel. Here we in fact do not have any kernel, we just have "normal" dot product, thus in 2d our decision boundary is always line.

$$K(X, Y) = X * Y$$

4.2.2 Gaussian kernel

As we know that kernel function shows similarity between two points. Now by looking on the formula of Gaussian kernel we can see that when $X \to Y$ then $K(X, Y) \to 1$

If X is far away from Y, then $\|X - Y\|^2 \to \infty$, and $K(X, Y) \to 0$



$$K(X, Y) = exp(-\frac{\|x - y\|^2}{2\sigma^2})$$

So here we can see that as A, B are near to each other their Gaussian kernel function which sometimes is called Radial Basis function. will give higher value, and thus B can have much more impact on A, then for example point C which is far from A.

There also several types of kernel such as

Polynomial kernel: It is popular in image processing

$$K(X, Y) = (X * Y + 1)^d$$

Sigmoid kernel: It is used as proxy for Neural Networks

$$K(X, Y) = tanh(\alpha X^T Y + c)$$

Conclusion

Support Vector Machine has become a very popular algorithm nowadays and is used for both classification and regression problems in many different spheres. SVM can not only make the reliable prediction but also can reduce redundant information. The SVM results are highly

competitive with those obtained by other approaches. In the sections above we discussed various SVM algorithms, which were Hard Margin, Soft Margin and Kernel-trick algorithms. Hard margin is used when the data is linearly separable, but as it is quite impossible in real life situations, Soft Margin is used to separate linearly non-separable data by a line - hyperplane allowing some error, which is, some data points can be on the wrong side of margin or on the wrong side of the hyperplane. On the other Hand kernel trick is used to separate non-linearly separable data using a different approaches. Thus, SVMs can be considered to be really important invention for nowadays and for future development of data science.

Evaluation

This paper was written during four weeks. The first few days we mainly devoted to research. Our goal was to find valid and reliable resources about different SVM algorithms and their applications. Individual researches were done mostly at home. However, we often had meetings at AUA, in order to do team discussions. Later, due to done researches during the first weeks, we came up with many questions concerning to different mathematical equations involved in solving problems using SVMs, so we gathered and tried to solve the problems together. Due to these group discussions, we were able to fix our mistakes and make our paper better both academically and structurally. Overall, our group managed to develop a well-constructed and thorough work regarding to Support Vector machine methods.

Appendix

Support Vector Classifier (Matlab)

```
clear all;close all;
% Generating the data
x = [randn(20,2);randn(20,2)+4];
y = [repmat(-1,20,1);ones(20,1)];
% Adding a bad point :)
x = [x;2 1];
y = [y;1];
% Plotting the data
types = {'ko','ks'};
fc = {[0 0 0],[1 1 1]};
val = unique(y);
ind = find(y==val(1));
figure(1); hold off
for i = 1:length(val)
   ind = find(y==val(i));
plot(x(ind,1),x(ind,2),types{i},'markerfacecolor',fc{i});
     hold on
end

% Setting up the optimization problem
N = size(x,1);
K = x*x';
H = (y*y').*K + 1e-5*eye(N);
f = ones(N,1);
A = [];b = [];
LB = zeros(N,1); UB = inf(N,1);
Aeq = y';beq = 0;
warning off
%Different values of Regularization parameters
Cvals = [10 5 2 1 0.5 0.1 0.05 0.01];

for cv = 1:length(Cvals)
   UB = repmat(Cvals(cv),N,1);
   % Following line runs the SVM
   alpha = quadprog(H,-f,A,b,Aeq,beq,LB,UB);
   % Compute the bias
   fout = sum(repmat(alpha.*y,1,N).*K,1)';
   ind = find(alpha>1e-6);
   bias = mean(y(ind)-fout(ind));

   %Plot the data, decision boundary and Support vectors
   figure; hold off
   ind = find(alpha>1e-6);

plot(x(ind,1),x(ind,2),'ko','markersize',15,'markerfacecolor',
[0.6 0.6 0.6],...
      'markeredgecolor',[0.6 0.6 0.6]);
   hold on
   for i = 1:length(val)
      ind = find(y==val(i));

plot(x(ind,1),x(ind,2),types{i},'markerfacecolor',fc{i});
   end

   xp = xlim;
   yl = ylim;
   % Because this is a linear SVM, we can compute w and
plot the decision
   % boundary exactly.
   w = sum(repmat(alpha.*y,1,2).*x,1)';
   yp = -(bias + w(1)*xp)/w(2);
   plot(xp,yp,'k','linewidth',2);
   ylim(yl);
   ti = sprintf('C: %g',Cvals(cv));
   title(ti);
end
%In the end you will get different figures with different
regularization
%parameters. You can see that as Cvals is increasing the
strictness of
%our line is increasing. If Cvals tend to infinity then we
will get
%hard margin classifier.
```

Hard Margin Classifier (Python)

```
import numpy as np
import matplotlib.pyplot as plt
import math
from sklearn.datasets.samples_generator import make_blobs
(X,y) =  make_blobs(n_samples=100,n_features=2,centers=2,cluster_std=1.4,random_state=40)
final_data = np.c_[np.ones((X.shape[0])),X] #Adding ones from the right to X for Thetta0 or b
plt.scatter(final_data[:,1],final_data[:,2],marker='o',c=y)
plt.axis([-5,10,-12,-1])
plt.show()
# So, now we should divide our values of y into two lists, if Value of y = 0 then it goes to negatives
#, else to positives
postives =[]
negatives=[]
for index,value in enumerate(y):
   if value==0:
      negatives.append(X[index])
   else:
      postives.append(X[index])

#Creating dictionary where keys are -1 or 1, and indexes which are equal to one of these cases
#are collected in one array for each case. That's why we use Python :)
data = {-1:np.array(negatives), 1:np.array(postives)}
feature_max=np.max(data[1])
feature_min=np.min(data[-1])
   alphas = [feature_max * 0.1,
             feature_max * 0.01, feature_max * 0.001,]
# This is the vector of step sizes
#all the required variables
w=[] #weights 2 dimensional vector
b=[] #bias
# As we step down our w vector, we'll test that vector in our constraint function, finding the largest b,
# if any, that will satisfy the equation, and then we'll store all of that data in our optimization dictionary.
def Optimizing_SVM(data):
   i=1
   global w
   global b
   dopt = {}
   #Next, we begin building an optimization dictionary as dopt, which will contain all optimizedvalues.
   #As we step down our w vector, we'll test that vector in our constraint function, finding the largest b,
    #that will satisfy the equation, and then we'll store all of that data in our optimization dictionary(dopt).
   #The dictionary will be in the following form { ||w|| : [w,b] }.
   transforms = [[1,1],[-1,1],[-1,-1],[1,-1]]
   b_step_size = 2
   b_multiple = 5
   w_optimum = feature_max*0.5
for alpha in alphas:
 w = np.array([w_optimum,w_optimum])
      optimized = False
      while not optimized:
         #b=[-maxvalue to maxvalue] we want to maximize b
         for b in np.arange(-1*(feature_max*b_step_size), feature_max*b_step_size, alpha*b_multiple):
              for transformation in transforms:  # we should check every version of the vector possible.
                w_t = w*transformation
                 correctly_classified = True
```

```
                # every data point should be correct
                for j in data:
                    for x in data[j]:
                        if j*(np.dot(w_t,x)+b) < 1:  # we want  yi*(np.dot(w_t,xi)+b) >= 1 for correct classification
                            correctly_classified = False
                            if correctly_classified:
                        dopt[np.linalg.norm(w_t)] = [w_t,b] #store w, b for minimum magnitude
            if w[0] < 0:
                optimized = True
            else:
                w = w - alpha
    norms = sorted([n for n in dopt])
        min_w = dopt[norms[0]] # Taking the key with the least value
        w = min_w[0]
        b = min_w[1]

        w_optimum = w[0]+alpha*2
Optimizing_SVM(data)
def hyperplane_value(x,w,b,v):
    return (-w[0]*x-b+v) / w[1]
def visualize(data):
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    plt.scatter(final_data[:,1],final_data[:,2],marker='o', c = y)
 # (w.x+b) = 1
    # positive support vector hyperplane
    psv1 = hyperplane_value(feature_min, w, b, 1)
    psv2 = hyperplane_value(feature_max, w, b, 1)
    ax.plot([feature_min,feature_max],[psv1,psv2], 'k')

    # (w.x+b) = -1
    # negative support vector hyperplane
    nsv1 = hyperplane_value(feature_min, w, b, -1)
    nsv2 = hyperplane_value(feature_max, w, b, -1)
    ax.plot([feature_min,feature_max],[nsv1,nsv2], 'k')

#     # (w.x+b) = 0
#     # Constructing the maximal margin
    mm1 = hyperplane_value(feature_min, w, b, 0)
    mm2 = hyperplane_value(feature_max, w, b, 0)
    ax.plot([feature_min,feature_max],[mm1,mm2], 'y--')

    plt.axis([-5,10,-12,-1])
    plt.show()
visualize(data)
```

Bibliography

1) Steinward, I., & CHristmann, A. (n.d.). *Support Vector Machines*.
2) James, G., & Witten, D. (n.d.). *An Introduction to Statistical Learning with Applications in R*.
3) Li, L. (n.d.). *Selected Applications of Convex Optimization*.
4) Vladimir Vapnik | Unlocking a Complex World Mathematically. (n.d.).
5) "Department of Computer Science." Royal Holloway, University of London. (n.d.).
6) "EECS at UC Berkeley." Gödel and God. (n.d.).
7) "Isabelle Guyon, PhD, FACMI." The Standards Standard | AMIA. (n.d.)

Image Sources

1) (n.d.). Retrieved from https://eight2late.wordpress.com/2017/02/07/a-gentle-introduction-to-support-vector-machines-using-r/
2) (n.d.). Retrieved from http://www.saedsayad.com/support_vector_machine.htm