Overview of Moderm AI and ML **Chapter 5 - (V0.1)**

Support Vector Machines (SVM)

C.V Jawahar

5.1 SVM Problem

We now know linear classifiers such as perceptron and logistic regression. We know that if the data is linearly separable, perceptron algorithm will converge with a feasible solution. i..e, a separating hyper plane. However, we know that not all separating hyper planes are equally useful. For example, a plane that "just" classifies a training sample is not the best. Intuitively this leaves higher chance for a test sample (even if it is somewhat similar to the training one) to get misclassified. Conceptually we prefer a separating hyper plane that is far from all the samples.

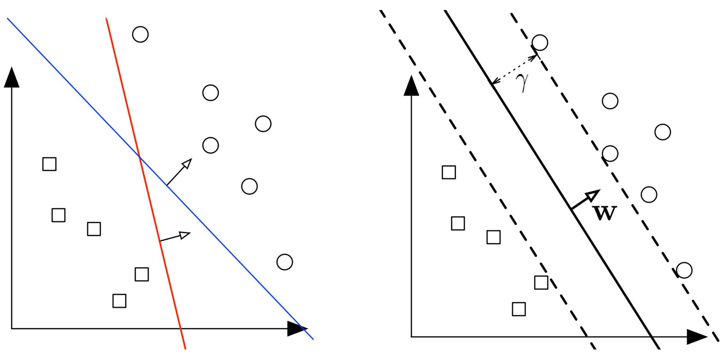

Figure 5.1: (a) Not all separating planes are equally useful. (b) The max-margin hyperplane and the side-planes.

In short, we want to find a separating hyperplane that maximimzes the margin. That is what support vector machines (SVM) are. SVMs are very popular classifiers even today. They have many nice theoretical properties. The optimization problem is convex and that is a special advantage.

We know from the mid school mathematics that the distance from origin to the line $ax + by + c = 0$ is $\frac{c}{\sqrt{a^2+b^2}}$. In a similar manner we can see that distance from origin to $\mathbf{w}^T\mathbf{x} + b = 1$ is $\frac{1-b}{||\mathbf{w}||}$. Similarly the distance from origin to the $\mathbf{w}^T\mathbf{x} + b = -1$ is $\frac{-1-b}{||\mathbf{w}||}$. Or the distace between the two side planes is $\frac{2}{||\mathbf{w}||}$.

Thus our objective is to maximize the margin or maximize $\frac{1}{||\mathbf{w}||}$ or minimize $\frac{1}{2}\mathbf{w}^T\mathbf{w}$.

Indeed the unconstrained minimization of this could lead to $\mathbf{w}$ becoming zero. That is not useful. Also this problem does not say anything about the samples correctly classified, which is our primary interest. Therefore, we need to add constraints to the optimization problem that says that the samples are correctly classified.

Let us assume that the samples are $(\mathbf{x}_i, y_i)\}$ $i = 1, \dots N$ where y_i is either $+1$ or -1.

- When $y_i = +1$ we would like the samples to be $\mathbf{w}^T\mathbf{x} + b \geq +1$
- When $y_i = -1$ we would like the samples to be $\mathbf{w}^T\mathbf{x} + b \leq -1$.

- We can combine these two constraints into one by multiplying y_i on both side. (Note that when y_i is -1 the inequality sign also reverses.),

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad \forall i$$

5.1.1 Hard Margin SVM Problem

The SVM problem is therefore

$$minimize \ \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

such that

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad \forall i$$
$$y_i \in \{-1, +1\}$$

Note that we made an assumption that the samples are linearly separable always. (otherwise, not all the constraints above can be satisfied.). This is not very practical. When we get a problem, we will not know whether the samples are linearly separable or not.

5.1.2 Soft Margin SVM Problem

We made a strong assumption that the samples are linearly separable. That is too restrictive in practice. Let us relax this by allowing a penalty ξ_i, when the constraint is violated. In a very similar manner to the hard margin formulation, we can write:

- When $y_i = +1$ we would like the samples to be $\mathbf{w}^T\mathbf{x} \geq +1 - \xi_i$
- When $y_i = -1$ we would like the samples to be $\mathbf{w}^T\mathbf{x} \leq -1 + \xi_i$.
- We can combine these two constraints into one by multiplying y_i on both side. (like for hard margin SVM) i.e.,

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \quad \forall i$$

If we allow ξ_is to be arbitrarily high, all samples can violate by large amount and the margin can be very high. However, that is not a very useful solution. We wish ξ_is are not required for all the samples.

Indeed if ξ_is are all zero, we will have our hard margin SVM that we saw already. Our new problem of interest is now to minimize both $\mathbf{w}^T\mathbf{w}$ and $\sum_{i=1}^{N}\xi_i$. There are two quantities to simultaneously minimize. We balance the relative importance of these two terms with a non-negative constant C. Our problem is now

$$\min \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\xi_i$$

such that

$$y_i(\mathbf{w}^T\mathbf{x}_i) \geq 1 - \xi_i \quad \forall i$$

The parameter C is possibly the only “tunable” parameter that SVMs have. If C is too small (say zero), we are allowing easy violations. i.e., the algorithm will look for large ξ_i and large margin, by discarding the concern of violations in the separability. If C is too large, then violations are taken too seriously and not the margin. The parameter C is one that one may have to set manually in the SVM implementations. Some of the libraries, you can vary C over a wide range and pick the best with an external loop.

5.2 Dual Problem and Solution

However, the popular problem is a *dual* version of the same. (since problem is convex, the optima of the primal and dual will be the same or duality gap will be zero.). With no derivation, let us write the dual problem as

$$maximize \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \tag{5.1}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

with $\alpha_i \geq 0$.

Here α_i are the Lagrangian multipliers. (though popular notation for Lagrangian multipliers is λ, SVMs use α.)

How did we get this dual problem? A brief explanation is given at the end of this lecture.

Dual problem is a classical quadratic programming problem. Many optimization libraries could help in this regard. Let us not aim for writing our own code for this at this stage.

The related $\mathbf{w}$ is:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$$

Q: How do you find b?

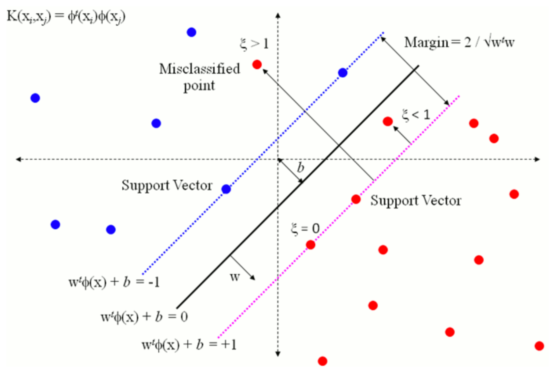

Figure 5.2: Soft Margin SVM setting. Please wait until the Kernels to apreciate why $\phi(x)$ come into the picture instead of $\mathbf{x}$. Figure from online resources.

5.3 Practice of SVMs

We know how to implement many of the algorithms that we studied. However, SVMs are not that easy in practice. There are nice implementations like libsvm, liblinear etc. and most of the popular libraries have very good implementations.

Roughly this is what happens:

- Input $(\mathbf{x}_i, y_i)$ for $i = 1, \dots, N$.
- Solve a quadratic optimization problem, i.e., the dual problem of SVM. Return non-zero α_i or the lagrangians corresponding to the support vectors.
- Given a test sample, compute the class label as $sign(\sum_{i \in SV} \alpha_i y_i \mathbf{x_i}^T \mathbf{x}) + b$.

There are also nice gradient descent solvers for SVMs (read pegasos algorithm, which is also extended for nonliner and softmargin).

5.4 Discussions

If we had removed some of the training samples, those are away from the side planes, the solution will not change. (why?) The solution depends only on the samples that are hard to classify i.e., the samples on the side planes.

When we solve the dual problem, the αs are sparse. i.e., only some samples have impact on the final solution.

Support Vectors Support vectors are the ones where α_i is non zero.

At the test time, we just need to test the sign of $\mathbf{w}^T \mathbf{x} + b$ and decide whether it is positive or negative class. i.e., decide as

$$sign(\sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b) \tag{5.2}$$

Dot Products Everywhere Another important thing to note is that the samples appear only as dot product in both training (the optimization problem equation 5.1) and the testing (equation 5.2). This is very important and we exploit this smartly when we extend the linear SVMs to nonlinear SVMs using Kernels.

Overview of Moderm AI and ML **Chapter 6 - (V0.1)**

Kernels

C.V Jawahar

6.1 Kernels and Feature Maps

We had seen the idea of dimensionality reduction in the past. We had seen PCA. We can also have many other dimensionality transformation techniques. The idea is that with an appropriate feature transformation the problem becomes "simpler" and the classifier/algorithm can do superior. Why don't we increase the dimension if that makes the problem simpler?

Consider a feature transformation (or feature map) as:

$$\mathbf{p} \to \phi(\mathbf{p}).$$

Let us start with a specific example: Let $\mathbf{p} = [p_1, p_2]^T$, and $\phi(\mathbf{p})$ be $[p_1^2, p_2^2, \sqrt{2}p_1p_2]$. You may wonder how this helps us or why we do this. Wait. We will see the utility as we move forward. Note the notations. Here the sample $\mathbf{p}$ has two features/dimensions p_1 and p_2.

A number of algorithms in machine learning use dot product as the basic operation. You already had seen $\mathbf{w}^T\mathbf{x}$. The beauty of dot product is that the result is scalar independent of the dimensionality of the vector. i.e., the dot product of two vectors in 2D and 3D will be still a scalar, independent of the dimension. Let us now compute the dot product of $\phi(\mathbf{p})$ and $\phi(\mathbf{q})$. Here $\mathbf{q}$ is also a 2D vector similar to $\mathbf{p}$. i.e., $\mathbf{q} = [q_1, q_2]^T$. We know that $\mathbf{p}^T\mathbf{q} = p_1q_1 + p_2q_2$. Let us compute the dot/scalar/inner product in the new feature space.

$$\begin{aligned}\phi(\mathbf{p})^T\phi(\mathbf{q}) &= [p_1^2, p_2^2, \sqrt{2}p_1p_2]^T[q_1^2, q_2^2, \sqrt{2}q_1q_2] \\ &= p_1^2q_1^2 + p_2^2q_2^2 + 2p_1p_2q_1q_2 \\ &= (p_1q_1 + p_2q_2)^2 \\ &= (\mathbf{p}^T\mathbf{q})^2 = \kappa(\mathbf{p}, \mathbf{q})\end{aligned}$$

What it says is something simple. If we want to compute the dot products of $\phi(\mathbf{p})$ and $\phi(\mathbf{q})$, what we need to do is only computing dot product of $\mathbf{p}$ and $\mathbf{q}$ and then square it. (indeed, that is true only for this specific $\phi()$) Note that if we compute $\phi(\mathbf{p})^T\phi(\mathbf{q})$ like this, we do not have to really compute $\phi()$ explicitly. That could be a huge advantage in many situations. Later today, we also would like to map $\mathbf{p}$ into infinite dimension with a $\phi()$. The advantage of not requiring to compute $\phi()$ will be a big advantage then.

Feature Map: Here, $\phi()$ is popularly called as a feature map.

Kernels: The function $\kappa()$ is a kernel function.

We saw the kernel function of $(\mathbf{p}^T\mathbf{q})^2$. This need not be square. It could be $(\mathbf{p}^T\mathbf{q})^d$. This is a polynomial kernel. With some effort we can write the $\phi()$ corresponding to this kernel. There are many other potential kernels. A kernel functions allows us to compute an inner/dot product in a new feature space.

Let us consider another $\phi()$ for the same $\mathbf{p} = [p_1, p_2]^T$. Let $\phi(\mathbf{p})$ as $[p_1^2, p_2^2, p_1p_2, p_2p_1]$. Here $\phi()$ maps from 2D to 4D. (instead of 2 to 3 as in the previous example).

$$\begin{aligned}\phi(\mathbf{p})^T\phi(\mathbf{q}) &= [p_1^2, p_2^2, p_1p_2, p_1p_2]^T[q_1^2, q_2^2, q_1q_2, q_2q_1] \\ &= p_1^2q_1^2 + p_2^2q_2^2 + p_1p_2q_1q_2 + p_1p_2q_1q_2 \\ &= (p_1q_1 + p_2q_2)^2 \\ &= (\mathbf{p}^T\mathbf{q})^2 = \kappa(\mathbf{p}, \mathbf{q})\end{aligned}$$

There can be many such kernels functions and feature maps. The above ones were only some examples. Does it mean that for any $\kappa(,)$ we have a corresponding $\phi()$? Can any function be a kernel function? There are many such curious questions for investigating later.

Q: What about a kernel like $(\mathbf{p}^T\mathbf{q})^2 + (\mathbf{p}^T\mathbf{q})^3$? What will be the corresponding feature map?

6.1.1 Motivation from Separability

Consider a 2D pattern of two concentric circles (figure missing). Inner circle is class 1 and outer circle from class 2. Consider a feature map of the form

$$\phi(\mathbf{p}) = \phi([p_1, p_2]^T) = [r, \theta]^T$$

where (r, θ) is the polar coordinates. It is simple to see that the concentric circles will become separable in the polar coordinates. (indeed only r could have been enough.)

We can also consider

$$\phi(\mathbf{p}) = \phi([p_1, p_2]^T) = [p_1^2, p_2^2]^T$$

What happens to the concentric circles?

The point to note is that a 'good' $\phi()$ is going to make our (classification) problem simpler (say linearly separable). There are more unanswered questions now. How do we find the useful $\phi()$ or $\kappa()$ for a new problem.

6.2 Kernel Matrix

For many problems we have N sample vectors and we need to compute all kernel values for all pairs. Popularly, a kernel matrix $\mathbf{K}$ with (i, j) th element as $\kappa(\mathbf{x}_i, \mathbf{x}_j)$.

Q: What are the properties of the kernel matrix? square? symmetric? what more?

6.3 Nonlinear/Kernel SVMs

The notion of Kernels is very nicely related to SVMs. One may wonder whether SVMs are designed for Kernels or Kernels are designed for SVMs!!.

Let us now come back to our SVM problem (dual).

$$maximize \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j \tag{6.3}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

with $\alpha_i \geq 0$. Solving this gives us a linear SVM.

Assume the input data $\{(\mathbf{x}_i, y_i)\}$ was mapped by $\phi()$, leading to $\{(\phi(\mathbf{x}_i), y_i)\}$. We can find a linear boundary in the new feature space. And the corresponding decision boundary in the original space is going to be a nonlinear one.

This makes the SVM problem (with appropriate constraints) as

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

or

$$max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) \tag{6.4}$$

Many practical solvers use the precomputed $N \times N$ kernel matrix. This avoids repeated computation of kernel functions. But this necessitates the need to store and manipulate a $N \times N$ matrix while solving. Not very nice for big data sets.

6.3.1 Training and Testing

When you solve this nonlinear SVM problem, we get α_i corresponding to the new nonlinear SVM. Or what we get is the nonlinear SVM in the original feature space. Typically the output of the training is a set of α_i (that are non zero).

At the test time, we only need to evaluate the sign of $\mathbf{w}^T \phi(\mathbf{x}) + b$. i.e.,

$$sign(\sum_{i=1}^{N} \alpha_i y_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) + b) \tag{6.5}$$

In a kernel setting this simply becomes:

$$sign(\sum_{i=1}^{N} \alpha_i y_i \kappa(\mathbf{x}_i, \mathbf{x}) + b) \tag{6.6}$$

Please note that α_i is sparse and we need to sum this only over the support vectors. However, at the test time, we need to have all the support vectors with us and the number of kernel evaluations is related to the number of support vectors. This makes the nonlinear SVMs slower at run time.

In the case of linear SVM, it is possible to compute $\mathbf{w}$ upfront and make each of the testing simple in computation. (Indeed while training, we still need to solve for α_i.).

Will the support vectors (and the magnitude of α_i) change with the choice of kernels? Yes. If you change the kernel, you need to solve the problem again and obtain the new α_i.

6.3.2 Example

Consider an example of XOR problem with $(-1,-1), -1; (-1,+1), +1; (+1,-1), +1; (+1,+1), -1$ Clearly the given four samples are not linearly separable. Assume we use a kernel $\phi(\mathbf{p}) = p_1 p_2$, the data becomes immediately separable. $+1, -1; -1, +1; -1, +1; +1 - 1$.

A quadratic kernel like $\kappa(\mathbf{p},\mathbf{q}) = (\mathbf{p}^T\mathbf{q}))^2$ or $(\mathbf{p}^T\mathbf{q}+1)^2$ will have a product term and this can make the XOR problem separable. (remember the example $\phi()$ we had at the beginning.)

6.4 Popular Kernels

We appreciate

- the utility of feature maps and kernels in"simplifying" the problem (eg. making the problem linearly separable).
- the fact that we do not have to evaluate the feature map $\phi()$ in practice. A small kernel computation in the original space is equivalent to the inner product in the feature space. An important point to note is that when we use kernels, we do not evaluate $\phi()$ or even do not have to know the explicit form of $\phi()$ itself. (Knowing $\phi()$ is still required for the exams!!)

The feature maps and the kernels that we saw is a toy kernel in 2D. We saw the $\phi()$ mapping frm 2D to 3D or 4D. Why not $\phi()$ mapping to an infinite dimensional space? Many popular kernels do that.!!

Many of the popular kernel functions

Linear Kernel	: $K(\mathbf{x},\mathbf{y}) = (\mathbf{x}\cdot\mathbf{y})$
Polynomial Kernel	: $K(\mathbf{x},\mathbf{y}) = (\mathbf{x}\cdot\mathbf{y})^d$
Radial Basis Kernel (RBF)	: $K(\mathbf{x},\mathbf{y}) = e^{-\gamma\|\mathbf{x}-\mathbf{y}\|^d}$
Sigmoid Kernel	: $K(\mathbf{x},\mathbf{y}) = tanh(\gamma(\mathbf{x}\cdot\mathbf{y}))$

We may not know the "right" kernel for a problem in our hand always. For example a simple product term worked for XOR. But a polynomial kernel, which has such a term is what we may use in practice. In practice we try multiple popular kernels (i.e., the ones that are in your library!!) and pick the one that gives us best results. RBF kernel is often preferred for many problems.

6.4.1 Kernels for more structured data

We can think of kernels as "similarity functions" that can be plugged into the machine learning algorithm. We had seen kernels for real vectors. In many problems the inputs could be objects like (i) strings, (ii) trees or (iii) graphs. Can kernels be used in such situations?

One can define kernel over strings, trees, graphs etc. There are many such kernels available in the literature. This makes it possible to treat objects like graphs the same way we do the vectors in vector spaces.

Now we can directly work on strings in SVMs. An SVM can train and test Strings (not just vectors) given that the appropriate kernels are used. This also makes the SVM more general and useful.