COMP9417 Kernel Methods Extended Solutions

Omar Ghattas

July 21, 2020

Additional comments for Questions 2 and 4 for the tutorial this week. Any errors are my own, and should be reported to omarghattas1991@gmail.com.

Extended Question 2

Recall that computing the SVM classifier is equivalent to solving the following constrained optimisation problem:

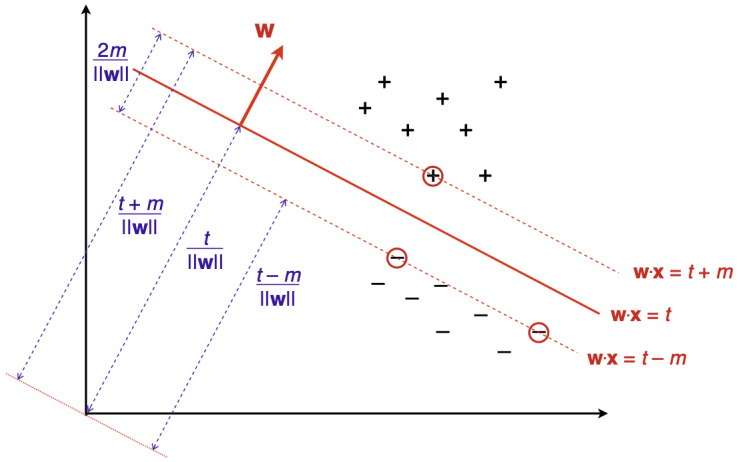
$$\arg\min_{\mathbf{w},t} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to } y_i(\mathbf{x}_i \cdot \mathbf{w} - t) \geq 1 \quad \text{for } i = 1, \ldots, n.$$

The way to interpret this expression is that we are looking for $\mathbf{w}, t$ such that:

1. We classify all points correctly by having observations from the first class being on one side of the line $\mathbf{x}_i \cdot \mathbf{w} = t + 1$ and points from the second class being on the opposite side of the line $\mathbf{x}_i \cdot \mathbf{w} = t - 1$. This is taken care of in the above expression by requiring $y_i(\mathbf{x}_i \cdot \mathbf{w} - t) \geq 1$ for all i, which captures both statements succintly. Compare this to the weaker condition required for perceptron learning: $y_i(\mathbf{x}_i \cdot \mathbf{w} - t) \geq 0$.

2. Require the margin between the two lines $\mathbf{x}_i \cdot \mathbf{w} = t \pm 1$ to be as 'fat' as possible. Recall that this margin has width $\frac{1}{\|\mathbf{w}\|}$, which we would like to maximise, or equivalently, minimise $\|\mathbf{w}\|$, or equivalently minimise $\frac{1}{2}\|\mathbf{w}\|^2$.

The plot below depicts the geometry of the problem (note that we take $m = 1$!).



Next, it was shown in the lecture that it is simpler to consider the dual problem:

$$\arg \max_{\alpha_1,\ldots,\alpha_n} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j(\mathbf{x}_i \cdot \mathbf{x}_j) \quad \text{subject to } \sum_{i=1}^{n}\alpha_i y_i = 0, \quad \alpha_i \geq 0 \quad \text{for } i = 1,\ldots,n.$$

In the initial problem, we learn the parameters $\mathbf{w}, t$, and in the dual problem, we focus instead on the vector $\alpha = (\alpha_1,\ldots,\alpha_n)^T$. Note that we can still recover the initial parameters from α through the relationship:

$$\mathbf{w} = \sum_{i=1}^{n}\alpha_i y_i \mathbf{x_i},$$

and for the parameter t, we can solve the equation $y_i(\mathbf{w}\cdot\mathbf{x}_i - t) = 1$ for any $\mathbf{x}_i$ on the decision boundary (any $\mathbf{x}_i$ that is a support vector), that is:

$$t = \mathbf{w}\cdot\mathbf{x}_i - \frac{1}{y_i}.$$

The following steps explain how to solve the dual problem for the simple dataset provided in the question.

Step 1. Considering the first term in the dual problem, we see that we will need the terms $\mathbf{x}_i \cdot \mathbf{x}_j$ for all i, j. The design matrix is the matrix of $\mathbf{x}_i$'s, defined by

$$X = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix} \in \mathbb{R}^{n\times p}$$

The Gram matrix (matrix of all pairwise dot products) is then $\mathbf{X}\mathbf{X}^T \in \mathbb{R}^{n\times n}$, in which case the (i,j)-th element of the Gram matrix gives us the term $\mathbf{x}_i \cdot \mathbf{x}_j$. Here however, we can note that we further need the terms terms $(y_i\mathbf{x}_i)\cdot(y_j\mathbf{x}_j)$, which we can get by defining the augmented design matrix

$$X' = \begin{bmatrix} \mathbf{x}_1^T y_1 \\ \mathbf{x}_2^T y_2 \\ \vdots \\ \mathbf{x}_n^T y_n \end{bmatrix} \in \mathbb{R}^{n\times p},$$

and the augmented Gram matrix $\mathbf{G}'$, as:

$$\mathbf{G}' \equiv (\mathbf{X}')(\mathbf{X}')^T = \begin{bmatrix} 10 & 5 & -3 \\ 5 & 5 & -1 \\ -3 & -1 & 1 \end{bmatrix}$$

This step is just to show that when computing the SVM, all we need is the matrix $\mathbf{G}$, and no longer are required to carry around $\mathbf{X}$. To see this more clearly, we can rewrite the dual problem as:

$$\arg\max_{\alpha_1,\ldots,\alpha_n} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j\mathbf{G}'[i,j] \quad \text{subject to } \sum_{i=1}^{n}\alpha_i y_i = 0, \quad \alpha_i \geq 0 \quad \text{for } i = 1,\ldots,n.$$

Step 2. The dual optimisation problem is thus

$$\arg\max_{\alpha_1,\alpha_2,\alpha_3} -\frac{1}{2}\left(10\alpha_1^2 + 10\alpha_1\alpha_2 - 6\alpha_1\alpha_3 + 5\alpha_2^2 - 2\alpha_2\alpha_3 + \alpha_3^2\right) + \alpha_1 + \alpha_2 + \alpha_3$$

subject to $\alpha_1 \geq 0, \alpha_2 \geq 0, \alpha_3 \geq 0$ and $\alpha_1+\alpha_2-\alpha_3 = 0$. Here, we note that since $\alpha_1+\alpha_2-\alpha_3 = 0 \implies \alpha_3 = (\alpha_1+\alpha_2)$, we can replace every occurence of α_3 accordingly and simplify our problem into a maximisation over α_1 and α_2 only:

$$\begin{aligned}&\arg\max_{\alpha_1,\alpha_2} -\frac{1}{2}\left(10\alpha_1^2 + 10\alpha_1\alpha_2 - 6\alpha_1(\alpha_1+\alpha_2) + 5\alpha_2^2 - 2\alpha_2(\alpha_1+\alpha_2) + (\alpha_1+\alpha_2)^2\right) + 2\alpha_1 + 2\alpha_2 \\ =&\arg\max_{\alpha_1,\alpha_2} -\frac{1}{2}\left(5\alpha_1^2 + 4\alpha_1\alpha_2 + 4\alpha_2^2\right) + 2\alpha_1 + 2\alpha_2\end{aligned}$$

Step 3. Compute the partial derivatives with respect to α_1, α_2:

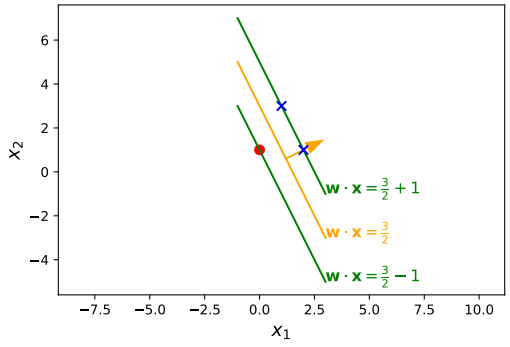
$$\frac{\partial}{\partial\alpha_1} = -5\alpha_1 - 2\alpha_2 + 2, \qquad \frac{\partial}{\partial\alpha_2} = -2\alpha_1 - 4\alpha_2 + 2$$

Step 4. Setting partial derivatives to zero and solving gives $\alpha_1 = \frac{1}{4}$ and $\alpha_2 = \frac{3}{8}$. Also, since $\alpha_3 = \alpha_1 + \alpha_2$ we have $\alpha_3 = \frac{5}{8}$. Note here that since $\alpha_i \neq 0$, all three points are support vectors, this is intuitive since our dataset is so small. In general, only the support vectors (points on the margins) will actually have a non-zero α_i term. In this sense, α_i captures the importance of the i-th point for learning the model. Note that points deep inside their respective classes are relatively unimportant, since if we are classifying points closest to the boundary between the classes correctly, we will always be classifying points deep in the class corectly too, and so their contribution to the model is zero.

Step 5. From slide 30: $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$ so

$$\begin{aligned}\mathbf{w} &= \frac{1}{4}\mathbf{x}_1 + \frac{3}{8}\mathbf{x}_2 - \frac{5}{8}\mathbf{x}_3 \\ &= \begin{bmatrix} 1 \\ 1/2 \end{bmatrix}\end{aligned}$$

Step 6. t can be obtained from any support vector, say $\mathbf{x}_3$, since $y_3(\mathbf{w} \cdot \mathbf{x}_3 - t) = 1$; this gives $t = \frac{3}{2}$. Note that in general, the support vectors are those points $\mathbf{x}_i$ for which $\alpha_i \neq 0$. We can now visualise the model we have learned:



.

As an extension, consider adding a point that is 'deep in the positive class', for example, $\mathbf{x}_4 = (10, 10)$, $y_4 = +1$. You should hopefully see that this will add zero information, and so we should get an identical model, and $\alpha_4 = 0$.

Step 7. Finally, the margin $1/\|\mathbf{w}\| = 1/\sqrt{(1^2 + (\frac{1}{2})^2)} = \sqrt{\frac{4}{5}} = \frac{2}{\sqrt{5}}$.

Extended Question 4

Before reading this, if you are confused about features representations and the kernel trick have a look through the additional comments that follow this. In this problem, we are told that the original space is $\mathbb{R}^2$, and we have two points:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix},$$

and further that we are using the kernel:

$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^2.$$

The goal is to figure out what feature representation (ϕ) we are choosing when we make use of this kernel. To figure this out, note that:

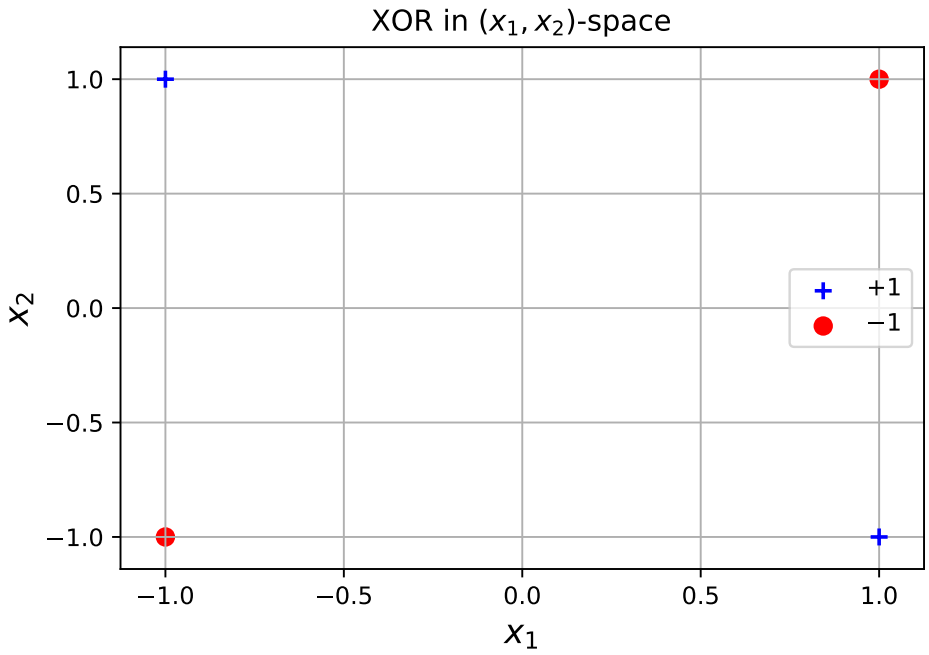
$$\begin{aligned} k(\mathbf{x}, \mathbf{y}) &= (\mathbf{x} \cdot \mathbf{y})^2 \\ &= \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \right)^2 \\ &= (x_1 y_1 + x_2 y_2)^2 \\ &= x_1^2 y_1^2 + x_2^2 y_2^2 + 2 x_1 y_1 x_2 y_2 \\ &= \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2} x_1 x_2 \end{bmatrix} \cdot \begin{bmatrix} y_1^2 \\ y_2^2 \\ \sqrt{2} y_1 y_2 \end{bmatrix} \\ &= \phi(\mathbf{x}) \cdot \phi(\mathbf{y}). \end{aligned}$$

So, using the kernel $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^2$ is equivalent to using the feature mapping:

$$\phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2} x_1 x_2 \end{bmatrix}.$$

Additional Comments on Feature Transformations

First, let's consider an example of non-linearly seperable data which we've seen before, namely the XOR function, which is the function depicted below:



.

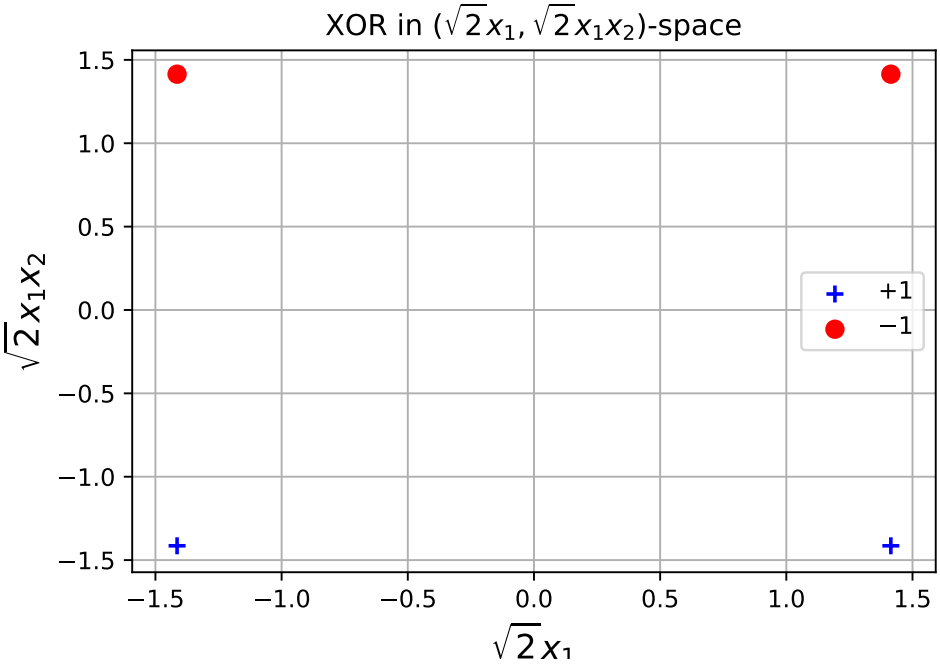
Now, clearly, a linear classifier will not work here! So, we can either try to learn a non-linear model, or, we could transform our data to try to make it linearly seperable. Consider the feature transformation:

$$\phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{bmatrix}.$$

In other words, ϕ is the function that takes a vector $[x_1, x_2]$ in 2-dimensions, and returns a 6-dimensional feature reprensetation. So, for our XOR problem, we would have:

$$\phi\left(\begin{bmatrix} 1 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ \sqrt{2} \\ \sqrt{2} \\ 1 \\ 1 \\ \sqrt{2} \end{bmatrix}, \quad \phi\left(\begin{bmatrix} -1 \\ -1 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ -\sqrt{2} \\ -\sqrt{2} \\ -1 \\ -1 \\ \sqrt{2} \end{bmatrix}, \quad \phi\left(\begin{bmatrix} -1 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ -\sqrt{2} \\ \sqrt{2} \\ 1 \\ 1 \\ -\sqrt{2} \end{bmatrix}, \quad \phi\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ \sqrt{2} \\ -\sqrt{2} \\ 1 \\ 1 \\ -\sqrt{2} \end{bmatrix}.$$

Now, we can't visualise 6-dimensional space, but let's pick 2 of the 6 features and plot, namely, we will plot in $(\sqrt{2}x_1, \sqrt{2}x_1x_2)$ space:



Clearly, a linear classifier will work now! Therefore, the idea is now to first transform your data using an appropriate feature transformation, ϕ, and then implement some of the algorithms we have been looking on the transformed data, for example, SVMs. Note that in SVM, the data enters the algorithm only through the dot product terms, $\mathbf{x}_i \cdot \mathbf{x}_j$. So, when we transform our data, we will instead have terms $\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$. This computation can get quite expensive, especially since we may be representing our points in a 1000, 10000, or even infinite dimensional feature space, and so computing the dot products now is computationally intensive/infeasible. This is where the idea of kernels comes in. A kernel function is a function $k(\mathbf{x}_i, \mathbf{x}_j)$ takes two points in the original space, and computes a value. It turns out that this value is the dot product in the **feature** space, i.e.:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi_k(\mathbf{x}_i) \cdot \phi_k(\mathbf{x}_j).$$

There are an infinite number of kernel functions to choose from, and the choice of kernel determines the representation ϕ_k for which we are choosing to represent our points, signified explicitly by the subscript k. Lets see an example using our above transformation of the XOR function. Let $\mathbf{x}, \mathbf{x}'$ be any two points in the original space, that is:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad \mathbf{x}' = \begin{bmatrix} x_1' \\ x_2' \end{bmatrix},$$

then, to use an algorithm like SVM, we need access to the terms

$$\phi(\mathbf{x}) \cdot \phi(\mathbf{x}') = \begin{bmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ \sqrt{2}x_1' \\ \sqrt{2}x_2' \\ x_1'^2 \\ x_2'^2 \\ \sqrt{2}x_1'x_2' \end{bmatrix} = 1 + 2x_1x_1' + 2x_2x_2' + x_1^2x_1'^2 + 2x_1x_1'x_2x_2'$$

which can be written simply as:

$$1 + 2x_1x_1' + 2x_2x_2' + x_1^2x_1'^2 + 2x_1x_1'x_2x_2' = (1 + \mathbf{x} \cdot \mathbf{x}')^2 = k(\mathbf{x}, \mathbf{x}').$$

In other words, instead of doing a long computation of dot products, all we need to do is compute the value of the kernel. This is called the polynomial kernel. So why is this so useful? Well, since we only need to compute dot products for many of these algorithms, we can get away with never having to explicitly define the feature transformation ϕ! We get it for free by choosing a kernel k. This is called the kernel trick, since it lets us compute a high dimensional dot product in the feature space simply by computing a dot product in the original lower dimensional space.

The polynomial kernel is neat, and let's us compute a 6-dimensional dot product via 2-dimensional dot products, but it doesn't show the full power of the kernel trick. Let's consider the Gaussian kernel, which projects data into **infinite** dimensional space. To see this, let $x_1, x_2 \in \mathbb{R}$, $\sigma > 0$, then:

$$\begin{aligned} k(x_1, x_2) &= \exp\left(-\frac{1}{2\sigma^2}(x_1 - x_2)^2\right) && \text{(Definition of the Gaussian kernel)} \\ &= \exp\left(-\frac{x_1^2}{2\sigma^2}\right)\exp\left(-\frac{x_2^2}{2\sigma^2}\right)\sum_{k=0}^{\infty}\frac{(x_1x_2)^k}{\sigma^{2k}k!} && \left(\text{Taylor expansion of } \exp\left(\frac{x_1x_2}{\sigma^2}\right)\right) \\ &= \exp\left(-\frac{x_1^2}{2\sigma^2}\right)\exp\left(-\frac{x_2^2}{2\sigma^2}\right)\sum_{k=0}^{\infty}\frac{x_1^k}{\sigma^k\sqrt{k!}}\frac{x_2^k}{\sigma^k\sqrt{k!}}, \end{aligned}$$

which can be written as $\phi(x_1) \cdot \phi(x_1)$ where:

$$\phi(x_i) = \exp\left(-\frac{x_i^2}{2\sigma^2}\right)\left[1, \quad \frac{x_i}{\sigma\sqrt{1!}}, \quad \frac{x_i^2}{\sigma^2\sqrt{2!}}, \quad \frac{x_i^3}{\sigma^3\sqrt{3!}}, \quad \dots\right]^T.$$

It is therefore impossible to explicitly represent $\phi(x)$ for the Gaussian case, and for many popular kernels. However, we never actually need to, since we can always calculate dot products in the feature space simply by evaluating k.