COMP9417 - Machine Learning
Tutorial: Kernel Methods

Question 1 (Dual Perceptron)

Review the development of the Perceptron training algorithm from lectures. Now compare this to the algorithm for Perceptron training *in dual form* introduced in the "Kernel Methods" lecture. The two algorithms are very similar but differ in a few crucial places. Provide an explanation of how the dual version of the algorithm relates to the original.

Solution:

For the purposes of this answer, we ignore the learning rate η, or equivalently assume that $\eta = 1$. In the original Perceptron training algorithm we try to classify the current example and check for an error, which occurs if $y_i\langle \mathbf{w}, \mathbf{x}_i\rangle \leq 0$. When this happens, we update the weight vector $\mathbf{w}$ by adding to it $\eta y_i \mathbf{x}_i$, i.e., $y_i \mathbf{x}_i$.

Thinking about this, clearly the weight vector that is learned will have been updated zero or more times for *every* example that has been misclassified. Denoting this number as α_i for example $\mathbf{x}_i$, the weight vector can be expressed as

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

On this *dual* view, we are learning instance weights α_i rather than feature weights w_j. An instance $\mathbf{x}$ is classified as

$$\hat{y} = \text{sgn}\left(\sum_{i=1}^{n} \alpha_i y_i \langle \mathbf{x}_i, \mathbf{x}\rangle\right)$$

During training, the only information needed about the training data is all pairwise dot products, which can be computed and stored easily in an $n \times n$ matrix called the Gram matrix:

$$\mathbf{G} = \mathbf{X}\mathbf{X}^T = \begin{bmatrix} \langle \mathbf{x}_1, \mathbf{x}_1\rangle & \langle \mathbf{x}_1, \mathbf{x}_2\rangle & \cdots & \langle \mathbf{x}_1, \mathbf{x}_n\rangle \\ \langle \mathbf{x}_2, \mathbf{x}_1\rangle & \langle \mathbf{x}_2, \mathbf{x}_2\rangle & \cdots & \langle \mathbf{x}_2, \mathbf{x}_n\rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \mathbf{x}_n, \mathbf{x}_1\rangle & \langle \mathbf{x}_n, \mathbf{x}_2\rangle & \cdots & \langle \mathbf{x}_n, \mathbf{x}_n\rangle \end{bmatrix}.$$

Question 2 (Feature Transformations)

Recall that the XOR function (graphed below) is not linearly separable. Show how we can learn the

XOR function using a linear classifier after applying a feature transformation to the original dataset. As a concrete example, show how to extend the Dual Perceptron from the previous question to learn the XOR function.

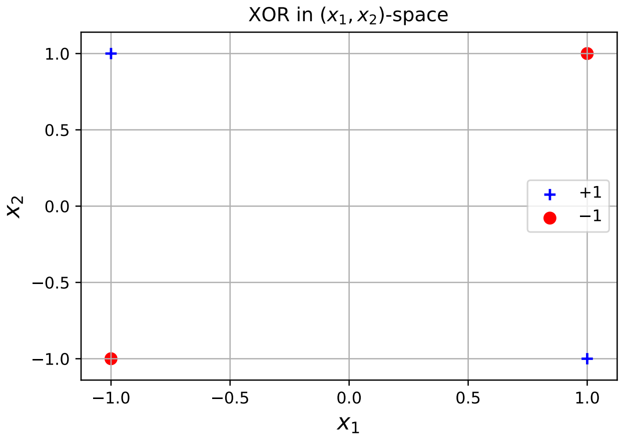


Figure 1: XOR function

Solution:

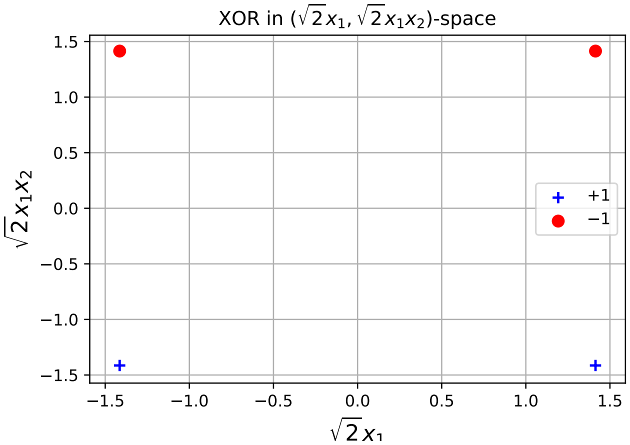
Clearly, a linear classifier trained on the original data will not work, since linear classifiers only work when the data is linearly sperable. We can either try to learn a non-linear model, or as suggested in the question, we could transform our data to try to make it linearly seperable. There are many possible transformations that will achieve this. Consider the following: transformation $\phi : \mathbb{R}^2 \to \mathbb{R}^6$ defined by

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

In other words, ϕ is the function that takes a vector $[x_1, x_2]$ in 2-dimensions (original space), and returns a 6-dimensional vector (feature space). 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 $(\sqrt{2}x_1, \sqrt{2}x_1x_2)$ (the second and sixth features respectively) space:



Clearly, a linear classifier will work now! Therefore, the idea is to first transform your data using an appropriate feature transformation, ϕ, and then implement some of the linear classifiers we have studied previously but on the transformed data. To learn the XOR function using the dual perceptron, we would simply run the dual perceptron on the transformed features, which means that we would only update the i-th weight, α_i if

$$\sum_{j=1}^{N} \alpha_j y_i y_j \langle \phi(\mathbf{x}_j), \phi(\mathbf{x}_i) \rangle < 0,$$

so all that changes is now we refer to the Gram matrix of the feature transformations:

$$\mathbf{\Phi} := \begin{bmatrix} \langle \phi(\mathbf{x}_1), \phi(\mathbf{x}_1) \rangle & \langle \phi(\mathbf{x}_1), \phi(\mathbf{x}_2) \rangle & \cdots & \langle \phi(\mathbf{x}_1), \phi(\mathbf{x}_n) \rangle \\ \langle \phi(\mathbf{x}_2), \phi(\mathbf{x}_1) \rangle & \langle \phi(\mathbf{x}_2), \phi(\mathbf{x}_2) \rangle & \cdots & \langle \phi(\mathbf{x}_2), \phi(\mathbf{x}_n) \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \phi(\mathbf{x}_n), \mathbf{x}_1 \rangle & \langle \phi(\mathbf{x}_n), \phi(\mathbf{x}_2) \rangle & \cdots & \langle \phi(\mathbf{x}_n), \phi(\mathbf{x}_n) \rangle \end{bmatrix}.$$

Note that we did not really need to use the entire feature transformation here, we could have simply transformed the data using the following transformation $\gamma : \mathbb{R}^2 \to \mathbb{R}^2$: defined by

$$\gamma(\mathbf{x}) = \gamma\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} \sqrt{2}x_1 \\ \sqrt{2}x_1x_2 \end{bmatrix},$$

which obviously also allows us to use a linear classifier (even without the $\sqrt{2}$) term. This is obviously a better choice since it is cheaper to compute a 2 dimensional feature transformation than a 6 dimensional one. We will explore in the next question why in general we use higher dimensional (even infinite dimensional) feature transformations.

Question 3. (The Kernel Trick)

Using the context of the previous question, discuss the computational issues that arise from having to compute high dimensional feature transformations. Show how these can be mitigated by using the Kernel trick, and use this to extend the dual perceptron learning to kernel perceptron learning.

Solution:

In the previous question, we saw that we could apply linear classifiers to non-linearly seperable data by first transforming them so that they are linearly separable in the feature space. We used the transformation ϕ, and showed that the dual perceptron can be trained on the transformed data. This requires us to compute a 6 dimensional transformations, and then compute all pairwise dot products between them - and in many cases we will have to compute very high dimensional feature transformations to get a linear classifier working on non linearly sperable data (think real world datasets) - and so this approach may become computationally prohibitive. This is where the idea of kernels comes in. A kernel function is a function $k : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$, which takes two points in the original (p-dimensional) space, and computes a number (a measure of similarity between the two inputs). There are an infinite number of kernel functions, here are a few popular ones:

- Polynomial Kernel: $k(\mathbf{x}_1, \mathbf{x}_2) = (m + \langle \mathbf{x}_1, \mathbf{x}_2 \rangle)^d$, where $m \geq 0$ and $d \in \mathbb{N}$ are hyper-parameters to be chosen.
- Gaussian Kernel: $k(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x}_1 - \mathbf{x}_2\|_2^2\right)$ with hyper-parameter $\sigma^2 > 0$.
- Laplace Kernel: $k(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(-\frac{1}{b}\|\mathbf{x}_1 - \mathbf{x}_2\|_1\right)$ with hyper-parameter $b > 0$.

Note that importantly, all the above kernels are functions of the data in the original space. Let's return to our feature transformation from the previous question. Let $\mathbf{x}, \mathbf{x}'$ be any two points in the original space, that is:

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

and recall that to run a linear classifier, we need all dot products of the transformed points, which are of the form:

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

which can be written simply as:

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

From this, it is clear that the dot product is in fact an evaluation of the polynomial kernel with $m = 1$ and $d = 2$, i.e. that

$$k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle.$$

Why is this so important? Well, instead of having to first compute $\phi(\mathbf{x})$, and then compute all pariwise dot products, the above calculation shows that we simply have to compute the kernel function between the two original vectors, and this computation is relatively much cheaper since it only operates on points in the original space! 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 works since recall that we only need access to dot prodcuts for learning, and we don't really need to compute the feature transformations explicitly, this is called the Kernel trick. In fact, we do not even have to think about computing a feature transformation function ϕ anymore, we simply choose a kernal, and each kernel gives us a corresponding feature transformation ϕ. This gives rise to the Kernel perceptron, which is just a renaming of the dual perceptron on transformed features defined above, so we update the i-th weight, α_i if

$$\sum_{j=1}^{N} \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) < 0,$$

so all that changes is now we refer to the Kernel matrix, which is the pairwise evaluation of the chosen kernel k on the original data:

$$\mathbf{K} := \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \cdots & k(\mathbf{x}_1, \mathbf{x}_n) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \cdots & k(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & k(\mathbf{x}_n, \mathbf{x}_2) & \cdots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix},$$

which is equivalent to $\mathbf{\Phi}$ in the previous question, but cheaper to compute relative to the naive approach of computing feateure transforations then taking dot products. Going back to the question of why we chose a 6 dimensional representation, it should be clear now that this was done so that we could relate back to the choice of kernel. In practice, if we are interested in polynomial features up to degree 8 say, then one would simply choose a polynomial kernel with $d = 8$.

Question 4. (Kernels and their Feature Representations)

In this question, we will show how the choice of kernel gives us different feature transformations. Note that in practice, we will simply choose a kernel and not be too concerned with the exact feature transformation, but it is important to know that different kernels correspond to different representations.

(a) Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^2$ (i.e. $\mathbf{x}$ and $\mathbf{y}$ are two dimensional vectors), and consider the kernel

$$k(\mathbf{x}, \mathbf{y}) = (2\langle \mathbf{x}, \mathbf{y} \rangle + 3)^3.$$

Compute the feature vector $\phi(\mathbf{x})$ correpsonding to this kernel. (In other words, the feature representation of $\mathbf{x}$ and $\mathbf{y}$ such that $\langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle = k(\mathbf{x}, \mathbf{y})$)

Solution:

Write $\mathbf{x} = (x_1, x_2)^T$ and $\mathbf{y} = (y_1, y_2)^T$ then

$$\begin{aligned} k(\mathbf{x}, \mathbf{y}) &= (2\langle \mathbf{x}, \mathbf{y}\rangle + 3)^3 \\ &\vdots \\ &= 8x_1^3y_1^3 + 8x_2^3y_2^3 + 24x_1^2y_1^2x_2y_2 + 24x_1y_1x_2^2y_2^2 \\ &+ 36x_1^2y_1^2 + 36x_2^2y_2^2 + 72x_1y_1x_2y_2 + 54x_1y_1 + 54x_2y_2 + 27 \end{aligned}$$

From here, it is easy to see that

$$\phi^T(x) = [\sqrt{8}x_1^3, \sqrt{8}x_2^3, \sqrt{24}x_1^2x_2, \sqrt{24}x_1x_2^2, \sqrt{36}x_1^2, \sqrt{36}x_2^2, \sqrt{72}x_1x_2, \sqrt{54}x_1, \sqrt{54}x_2, \sqrt{27}]$$

(b) **Challenge:** Let $x, y \in \mathbb{R}$, and consider the Gaussian kernel:

$$k(x, y) = \exp\left(-\frac{1}{2\sigma^2}(x-y)^2\right), \qquad \sigma^2 > 0.$$

Hint: Use a Taylor expansion to rewrite the exponential in terms of a summation.

Solution:
This exercise shows the full power of the kernel trick, since the Gaussian kernel allows us to compute **infinite** dimensional feature representations!

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

which can be written as $\langle \phi(x), \phi(y)\rangle$ where:

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

so we see that the implicit feature representation when using the Gaussian kernel is infinite dimensional! This means that by computing simple functions in the original space (via the kernel function), we are able to get powerful representations of the data without having to do any computation in infinite dimensional space!

Note again that in practice, we never actually need to compute explicit feature vectors, since we can always calculate dot products in the feature space simply by evaluating k, and for many kernels, such a computation is impossible anyway.

Question 5 (More of the Kernel Trick)

You are told that the "kernel trick" means that a non-linear mapping can be realised from the original data representation to a new, implicit feature space simply by defining a kernel function on dot products of pairs of instances from the original data. To see why this is so, you take two instances $\mathbf{x} = [1, 2]^T$ and $\mathbf{y} = [3, 2]^T$, and take their dot product $\langle \mathbf{x}, \mathbf{y} \rangle$ and obtain the answer 7. Clearly, raising this dot product to the power of two will give $(\langle \mathbf{x}, \mathbf{y} \rangle)^2 = 49$. Now expand out this expression to show that this is the same answer you would have obtained if you had simply done a set of feature transformations on the original data.

Solution:

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:

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

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:

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

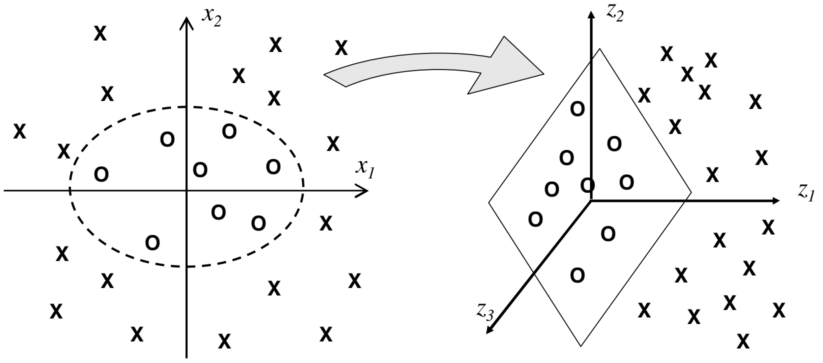
So, using the kernel $k(\mathbf{x}, \mathbf{y}) = (\langle \mathbf{x}, \mathbf{y} \rangle)^2$ is equivalent to using the feature mapping:

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

Question 6 (More Feature Transformations)

Consider the following depiction of a feature transformation from two dimensional space ($\mathbb{R}^2$) to three dimensional space ($\mathbb{R}^3$). Why would we use such a transformation? Generate one example from the ∘ class in the original space, and another example from the × class in the original space, and show their transformed values in the new space.

$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := \left(x_1^2, x_2^2, \sqrt{2}x_1x_2\right)$$



Solution:

The transformation allows us to use a linear classifier to correctly classify the data. In the original space, the points are not linearly separable and the only way to do classification here would be to use a non-linear model, which can be difficult. The smart choice of transformation allows us to use our existing tools for linear classification, such as the perceptron. We can demonstrate this transformation on two points, in the $\circ$ class, consider $x^\circ = (\sqrt{2}, \sqrt{2})$, and from the $\times$ class consider $x^\times = (2\sqrt{2}, 2\sqrt{2})$ then denote the transformation by ϕ, where

$$\phi : \mathbb{R}^2 \to \mathbb{R}^3, \qquad \phi(x_1, x_2) = (x_1^2, x_2^2, \sqrt{2}x_1x_2),$$

then we have

$$\phi(x^\circ) = (2, 2, 2\sqrt{2})$$
$$\phi(x^\times) = (8, 8, 8\sqrt{2}).$$

Question 7 (Support Vector Machines)

The Support Vector Machine is a essentially an approach to learning linear classifiers, but uses a alternative objective function to methods we looked at before, namely *maximising the margin*. Learning algorithms for this problem typically use quadratic optimization solvers, but it is possible to derive the solution manually for a small number of support vectors.

Here is a toy data set of three examples shown as the matrix $\mathbf{X}$, of which the first two are classified as positive and the third as negative, shown as the vector $\mathbf{y}$. Start by constructing the *Gram matrix* for this data, incorporating the class labels, i.e., form the matrix $\mathbf{X}'(\mathbf{X}')^T$. Then solve to find the support vectors, their Lagrange multipliers α, then determine the weight vector $\mathbf{w}$, threshold t and the margin m.

$$\mathbf{X} = \begin{bmatrix} 1 & 3 \\ 2 & 1 \\ 0 & 1 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} +1 \\ +1 \\ -1 \end{bmatrix} \qquad \mathbf{X}' = \begin{bmatrix} 1 & 3 \\ 2 & 1 \\ 0 & -1 \end{bmatrix}$$

To find a maximum margin classifier requires finding a solution for $\mathbf{w}$, t and margin m. For this we can use the following steps (refer to slides 30–35 from the "Kernel Methods" lecture):

1. Set up the Gram matrix for labelled data
2. Set up the expression to be minimised
3. Take partial derivatives
4. Set to zero and solve for each multiplier
5. Solve for $\mathbf{w}$
6. Solve for t
7. Solve for m

Solution:

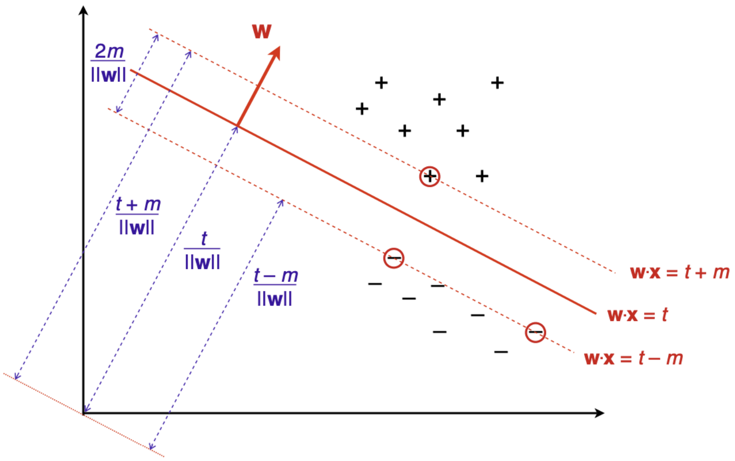
Recall that computing the SVM classifier is equivalent to solving the following constrained optimisation problem:

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

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 $\langle \mathbf{x}_i, \mathbf{w}\rangle = t + 1$ and points from the second class being on the opposite side of the line $\langle \mathbf{x}_i, \mathbf{w}\rangle = t - 1$. This is taken care of in the above expression by requiring $y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle - t) \geq 1$ for all i, which captures both statements succintly. Compare this to the weaker condition required for perceptron learning: $y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle - t) \geq 0$.
2. Require the margin between the two lines $\langle \mathbf{x}_i, \mathbf{w}\rangle = t \pm 1$ 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:

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

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 $y_i(\langle \mathbf{w}, \mathbf{x}_i\rangle - t) = 1$ for any $\mathbf{x}_i$ on the decision boundary (any $\mathbf{x}_i$ that is a support vector), that is:

$$t = \langle \mathbf{w}, \mathbf{x}_i\rangle - \frac{1}{y_i}.$$

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

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

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

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 $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$. Here however, we can note that we further need the terms terms $\langle y_i\mathbf{x}_i, y_j\mathbf{x}_j \rangle$, which we can get by defining the augmented design matrix

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

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:

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

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:

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

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

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

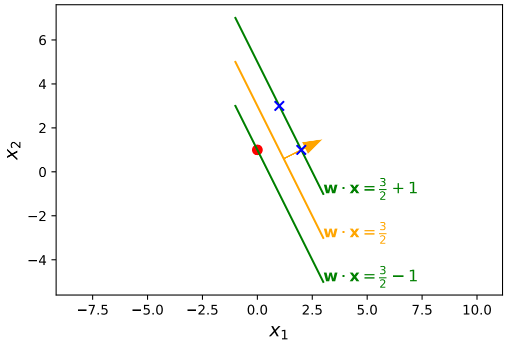
4. Setting partial derivatives to zero and solving gives $\alpha_1 = \frac{1}{4}$ and $\alpha_2 = \frac{3}{8}$. Also, since $\alpha_3 = \alpha_1 + \alpha_2$ we have $\alpha_3 = \frac{5}{8}$. Note here that since $\alpha_i \neq 0$, all three points are support vectors, this is intuitive since our dataset 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.

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

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

6. t can be obtained from any support vector, say $\mathbf{x}_3$, since $y_3(\langle \mathbf{w}, \mathbf{x}_3 \rangle - t) = 1$; 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 $\alpha_4 = 0$.

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