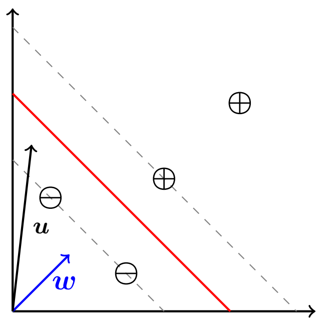
Support Vector Machines (SVMs)

The concept of **Support Vector Machines (SVMs)** was introduced by Vapnik and Chervonenkis in the 1990s. SVMs are binary classifiers, i. e. the data to be classified belongs to either of the two classes $\oplus$ (positive class) or $\ominus$ (negative class). The algorithm tries to determine a linear hyperplane such that the data is classified correctly. In general, there are many hyperplanes that achieve this (if the data is linearly separable). One reasonable choice as the best hyperplane is the one that represents the **largest separation**, or **margin**, between the two classes.



The central question is which decision rule do we have to use in order to separate the two classes in such a way? To answer this question, let us suppose we had a vector $\boldsymbol{w}$ which can be of any length and which is constrained to be **perpendicular to the hyperplane**. Also, let $\boldsymbol{u}$ be an unknown example which we wish to classify. In order to do so, we have to determine whether the unknown example is on the right or on the left side of the decision boundary (red line in the figure on the left). The projection of the vector $\boldsymbol{u}$ onto $\boldsymbol{w}$ provides an indication for that. The larger the projected value the more likely does the unknown example belong to the positive class. All we have to do is to compute $\boldsymbol{w}^\intercal \boldsymbol{u}$ and check whether this quantity is greater or equal to some constant c: $\boldsymbol{w}^\intercal \boldsymbol{u} \geq c$. Without loss of generality, we can say:

$$\boxed{\boldsymbol{w}^\intercal \boldsymbol{u} + b \geq 0} \tag{1}$$

where we have set $c = -b$. Equation (1) represents our **decision rule**. If we compute a value greater or equal to 0 for the unknown example $\boldsymbol{u}$, we will classify it as a positive example $\oplus$, otherwise we will assign the negative class $\ominus$.
Up to now $\boldsymbol{w}$ and b are unknown to us. All we know is that $\boldsymbol{w}$ has to be perpendicular to the decision boundary. Unfortunately, there are infinitely many possible solutions for $\boldsymbol{w}$ that satisfy this constraint because it can be of any length. $\Rightarrow$ **We need additional constraints!**

Suppose we were given a positive and a negative sample from the training set, denoted by $\boldsymbol{x}_\oplus$ and $\boldsymbol{x}_\ominus$, respectively. We insist that the following two equations hold true:

$$\boldsymbol{w}^\intercal \boldsymbol{x}_\oplus + b \geq 1 \tag{2}$$

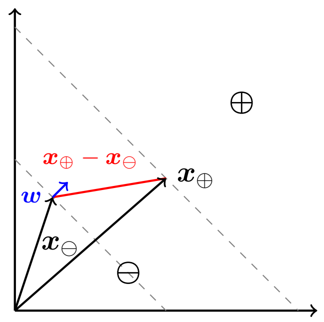
$$\boldsymbol{w}^\intercal \boldsymbol{x}_\ominus + b \leq -1 \tag{3}$$

For mathematical convenience, we will now condense equations (2) and (3) into a single formula. For that let us introduce a variable $y^{(i)}$, such that $y^{(i)} = 1$ if the i-th training sample belongs to the positive class, and $y^{(i)} = -1$ if it belongs to the negative class. You may think of $y^{(i)}$ as the class label. If we multiply equation (2) by $y^{(i)} = 1$, we get $y^{(i)}(\boldsymbol{w}^\intercal \boldsymbol{x}^{(i)} + b) \geq 1$. If we multiply equation (3) by $y^{(i)} = -1$, we get $y^{(i)}(\boldsymbol{w}^\intercal \boldsymbol{x}^{(i)} + b) \geq 1$. As we can see, the two equations turn out to be the same, therefore we are allowed to write:

$$y^{(i)}(\boldsymbol{w}^\intercal \boldsymbol{x} + b) - 1 \geq 0 \tag{4}$$

Furthermore, we insist that $y^{(i)}(\boldsymbol{w}^\intercal \boldsymbol{x} + b) - 1 = 0$ for all samples that lie on the margin boundaries (represented by the gray dashed lines in the figure above). Remember that the goal of the SVM algorithm is to find the hyperplane which represents the largest separation $\mathcal{S}$ between the two classes. Let us therefore define how this can be measured.

The separation $\mathcal{S}$ between the two classes can be computed by subtracting the positive example $\boldsymbol{x}_\oplus$ from the negative example $\boldsymbol{x}_\ominus$ (NB: Both vectors lie on the margin boundary) and projecting the result onto vector $\boldsymbol{w}$.



$$\mathcal{S} = (\boldsymbol{x}_\oplus - \boldsymbol{x}_\ominus)^\intercal \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} = \frac{\boldsymbol{x}_\oplus^\intercal \boldsymbol{w} - \boldsymbol{x}_\ominus^\intercal \boldsymbol{w}}{\|\boldsymbol{w}\|} \tag{5}$$

What can we get out of this? Equation (4) tells us that $\boldsymbol{x}_\oplus^\intercal \boldsymbol{w} = 1 - b$. Analogously, we find that $-\boldsymbol{x}_\ominus^\intercal \boldsymbol{w} = 1 + b$.[1] If we put these results back into equation (5), we notice that the separation $\mathcal{S} = (1 - b + 1 + b) \cdot \frac{1}{\|\boldsymbol{w}\|} = \frac{2}{\|\boldsymbol{w}\|}$. If we want the classes to separated as widely as possible we will have to maximize this quantity. Let us rewrite this optimization problem a little bit in order to make it mathematically more convenient:

$$\max \frac{2}{\|\boldsymbol{w}\|} \Leftrightarrow \max \frac{1}{\|\boldsymbol{w}\|} \qquad \text{drop constant} \tag{6}$$

$$\Leftrightarrow \min \|\boldsymbol{w}\| \qquad \text{minimize denominator to maximize a fraction} \tag{7}$$

$$\Leftrightarrow \min \frac{1}{2}\|\boldsymbol{w}\|^2 \qquad \text{this will be more convenient later on} \tag{8}$$

[1] We have equality because both vectors, $\boldsymbol{x}_\oplus$ and $\boldsymbol{x}_\ominus$, lie on the margin boundary and we insisted that for such vectors $y^{(i)}(\boldsymbol{w}^\intercal \boldsymbol{x} + b) - 1 = 0$.

But we cannot simply optimize this quantity without honoring our constraints which we specified earlier. What we get is a **constrained quadratic optimization problem**. How do we solve such problems? If we want to solve an optimization problem subject to constraints we will have to use so called **Lagrange multipliers** α_i. This procedure gives us a new expression which we can optimize without having to think of the constraints anymore. Please have a look at the mathematical foundations in case you do not know how Lagrange optimization works.

Equation (9) is the Lagrange function $\mathcal{L}$ which we have to optimize:[2]

$$\mathcal{L} = \frac{1}{2}\|\boldsymbol{w}\|^2 - \underbrace{\sum_{i=1}^{n} \alpha_i (y^{(i)}(\boldsymbol{w}^\intercal \boldsymbol{x}^{(i)} + b) - 1)}_{\text{constraints (one per training sample)}} \tag{9}$$

In the equation, we have a sum over all n training examples, i. e. we have one constraint for each element in the training data. Later we will see that the Lagrange multiplier α_i will be different from zero if $\boldsymbol{x}^{(i)}$ is a **support vector**.
As usual, we have to compute the derivatives with respect to all quantities which might vary – in our case we have to differentiate with respect to $\boldsymbol{w}$ and b – set the derivatives to zero and solve.

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{n} \alpha_i y^{(i)} \boldsymbol{x}^{(i)} = 0 \qquad \Rightarrow \boxed{\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y^{(i)} \boldsymbol{x}^{(i)}} \tag{10}$$

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum_{i=1}^{n} \alpha_i y^{(i)} = 0 \qquad \Rightarrow \boxed{\sum_{i=1}^{n} \alpha_i y^{(i)} = 0} \tag{11}$$

The result which equation (10) gives us is quite remarkable. It tells us that $\boldsymbol{w}$ is a **linear combination of the training data samples**. Let us put these results back into $\mathcal{L}$ (equation 9):

$$\begin{aligned}\mathcal{L} = &\frac{1}{2}\left(\sum_{i=1}^{n} \alpha_i y^{(i)} \boldsymbol{x}^{(i)}\right)^\intercal \left(\sum_{j=1}^{n} \alpha_j y^{(j)} \boldsymbol{x}^{(j)}\right) \\ &- \left(\sum_{i=1}^{n} \alpha_i y^{(i)} \boldsymbol{x}^{(i)}\right)^\intercal \left(\sum_{j=1}^{n} \alpha_j y^{(j)} \boldsymbol{x}^{(j)}\right) - \underbrace{\sum_{i=1}^{n} \alpha_i y^{(i)}}_{=0 \text{ (eq. 11)}} b + \sum_{i=1}^{n} \alpha_i\end{aligned}$$

[2]It does not matter whether the constraints are added or subtracted. The ultimate result remains unchanged, only the Lagrange multipliers will be different (but we don't care about this).

$$= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} \boldsymbol{x}^{(i)\intercal} \boldsymbol{x}^{(j)} \tag{12}$$

Equation (12) is called the **Wolfe dual**. This equation can be solved easily by standard optimization software. You may wonder why we did not optimize equation (9) directly. Why did we spend so much time on deriving the dual form of the problem? The answer is that, down the way, we have found many useful relationships, e. g. that $\boldsymbol{w}$ is a linear combination of the training data. Also note, that equation (12) is no longer dependent on $\boldsymbol{w}$ or b, but is expressed solely in terms of dot products between pairs of input data samples.
Finally, we may also rewrite equation (1) – the decision rule – as follows:

$$\sum_{i=1}^{n} \alpha_i y^{(i)} \boldsymbol{x}^{(i)\intercal} \boldsymbol{u} + b \geq 0 \tag{13}$$

This procedure works very well if the data is **linearly separable**. **Question:** *What if the data is not linearly separable?* SVMs are linear classifiers. However, we can apply a trick to model non-linear decision boundaries with SVMs. In machine learning we usually switch to a higher-dimensional space where the problem is more convenient so solve. We have already learned about the concept of basis functions $\varphi(\boldsymbol{x})$ as one possible approach. Therefore, we might consider replacing the dot product between pairs of training samples $\boldsymbol{x}^{(i)\intercal}\boldsymbol{x}^{(j)}$ with $\varphi(\boldsymbol{x}^{(i)})^\intercal \varphi(\boldsymbol{x}^{(j)})$. The disadvantage of this approach is that we explicitly have to compute the transformations prior to computing the dot product. We could be much more efficient if we had a function which directly gave us the value of the dot product **without having to compute the transformations first**. This is where the powerful concept of **kernels** comes into play. A kernel function $\mathcal{K}$ is defined as follows:

$$\mathcal{K}(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \varphi(\boldsymbol{x}^{(i)})^\intercal \varphi(\boldsymbol{x}^{(j)}) \tag{14}$$

One of the most popular kernel functions is the **Gaussian kernel**, also known as **radial basis function (RBF) kernel**:

$$\mathcal{K}(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \exp\left\{ \frac{\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\|}{\sigma} \right\} \tag{15}$$

The kernel concepts works well in practice. **However, we must be careful not to overfit the training data.** Also, too many data points will slow down the optimization process heavily. This makes SVMs less usable with large data sets ($> 50{,}000$ samples).

Source: This document is a transcription of Prof. Patrick Winston's lecture at MIT. Cf. `https://www.youtube.com/watch?v=_PwhiWxHK8o`.

Alternative approach:

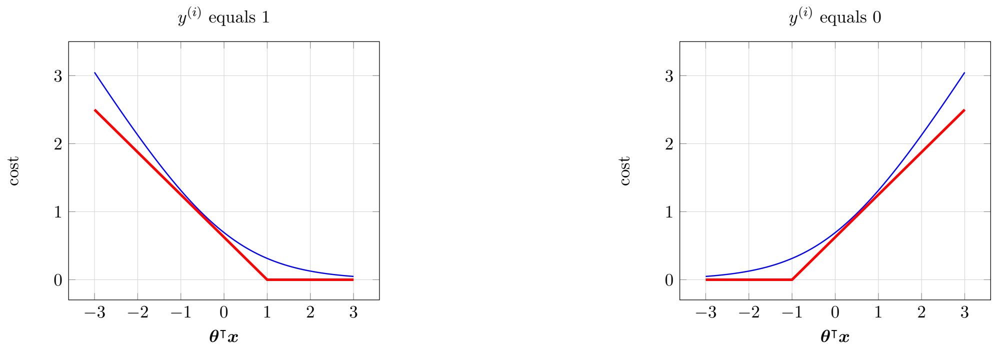
Alternatively, the SVM optimization problem can be derived starting from logistic regression. Recall that when using logistic regression, we optimize the **binary cross entropy loss function** which – for a single data point – is given by equation (16):

$$-\left(y^{(i)} \log h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) + (1 - y^{(i)}) \log(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}))\right) \tag{16}$$

where $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ is given by the **sigmoid function**:

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1 + \exp\{-\boldsymbol{\theta}^{\intercal}\boldsymbol{x}\}} \tag{17}$$

The two images below plot the binary cross entropy cost function (blue graph). The case $y^{(i)} = 1$ is depicted on the left hand side, while the case $y^{(i)} = 0$ is shown on the right hand side. In order to derive the SVM algorithm, we will now replace this cost function by another one which is represented by the red graph in the figure below. This cost function is generally referred to as **Hinge loss**.



Equation (16) computes the cost based on a single training example. The error of the model on the entire data set is calculated as presented in equation (18). We basically compute the loss for all training samples individually, and sum up all intermediate results. Additionally, we include a **regularization term** which is controlled by the hyper-parameter λ. In equation (19) we have replaced the original cost function (blue) with the new hinge loss function (red).

In order to be consistent with the SVM literature, we have replaced the hyper-parameter λ with another hyper-parameter called C. Setting $C = \frac{1}{\lambda}$ gives us the same result as in equation (19). Equation (20) shows the final error function for SVM optimization.

$$\mathcal{J}(\boldsymbol{\theta}) = \underbrace{\sum_{i=1}^{n} y^{(i)}(-\log h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})) + (1-y^{(i)})((-\log(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}))))}_{\text{binary cross entropy}} + \underbrace{\frac{\lambda}{2}\sum_{j=1}^{m}\theta_j^2}_{\text{regulariz.}} \tag{18}$$

$$= \sum_{i=1}^{n} y^{(i)} \cdot \text{cost}_1(\boldsymbol{\theta}^\intercal \boldsymbol{x}) + (1-y^{(i)}) \cdot \text{cost}_0(\boldsymbol{\theta}^\intercal \boldsymbol{x}) + \frac{\lambda}{2}\sum_{j=1}^{m}\theta_j^2 \tag{19}$$

$$= C \underbrace{\sum_{i=1}^{n} y^{(i)} \cdot \text{cost}_1(\boldsymbol{\theta}^\intercal \boldsymbol{x}) + (1-y^{(i)}) \cdot \text{cost}_0(\boldsymbol{\theta}^\intercal \boldsymbol{x})}_{(*)} + \frac{1}{2}\sum_{j=1}^{m}\theta_j^2 \tag{20}$$

Now suppose we would choose C to be a very large number (e. g. 100,000). The optimization algorithm would then try to minimize the first term (*) such that it eventually becomes 0. In order for this term to become zero, we will have to ensure that $\boldsymbol{\theta}^\intercal \boldsymbol{x}^{(i)} \geq 1$ if $y^{(i)} = 1$, and $\boldsymbol{\theta}^\intercal \boldsymbol{x}^{(i)} \leq -1$ if $y^{(i)} = 0$ (cf. plots above).

This means that for sufficiently large values of C, we can ignore the first term and instead optimize the subjective function $\frac{1}{2}\sum_{j=1}^{m}\theta_j^2 = \frac{1}{2}\|\boldsymbol{\theta}\|^2$ subject to the constraints $\boldsymbol{\theta}^\intercal \boldsymbol{x}^{(i)} \geq 1$ if $y^{(i)} = 1$, and $\boldsymbol{\theta}^\intercal \boldsymbol{x}^{(i)} \leq -1$ if $y^{(i)} = 0$. This is basically equivalent to the optimization problem given by equation (9). This means that SVMs can be derived starting from the logistic regression algorithm. **What we have also shown is that logistic regression can be adjusted in a way such that it becomes a large margin classifier which behaves like an SVM.** This can be achieved by using the hinge loss instead of the original loss function.

Source: Please refer to Andrew Ng's Coursera course for more information. In particular watch the two videos: `https://www.youtube.com/watch?v=hCOIMkcsm_g&list=PLNeKWBMsAzboNdqcm4YY9x7Z2s9n9q_Tb&index=1` and `https://www.youtube.com/watch?v=Ccje1EzrXBU&list=PLNeKWBMsAzboNdqcm4YY9x7Z2s9n9q_Tb&index=2`