TEST FOR DEV: Introduction to Data Science

WS 23/24, RWTH Aachen

January 22, 2024

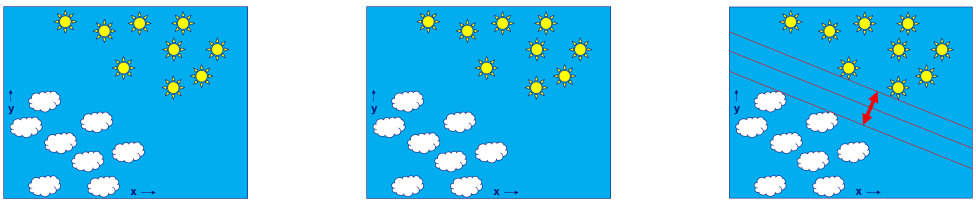
1 Support vector machine (SVM)

1.1 Basic functionality

Let's suppose we have the following dataset where

- Our target feature has two possible classes: sun and cloud.
- We have two descriptive features: x and y

The data set is separable, and our goal is now to find the best hyperplane (a line here) separating the two classes.



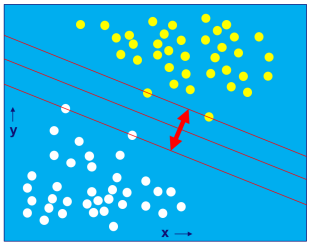
(a) Original data set (b) Proposal separating lines (c) Line with margin

Figure 1.1: Example for SVM

We need to determine which proposal lines are better than others. The idea to determine this is that the degrees of freedom become less when the separating hyperplane gets thicker. The best hyperplane is:

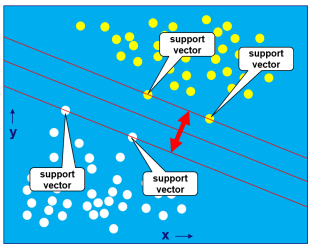
- The best-separating hyperplane is the one fitting between the data points with the maximal thickness (= safety margin).
- Therefore, SVMs also tend to be more robust than logistic regression.

Consider now a more abstract data set:



(a) Data set with hyperplane and safety margin

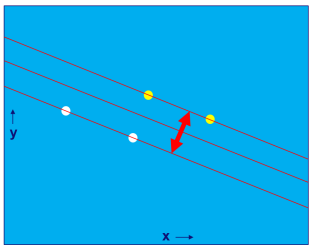
We have instances that are (atomic) points in an n-dimensional space.



(b) Support vectors

Support vector

Support vectors (SV) are the boundary points $\rightarrow$ only those matter.



(c) Reduced problem

Quadratic programming

Define a **QP** problem (quadratic programming) to make (compared to logistic regression):

- Things more efficient
- It less sensitive to outliers
- Reduce the risk of overfitting

But, there are applications where logistic regression works better.

Figure 1.2: SVM basic idea

The dimension of the separating hyperplane is always $n-1$ where n is the dimension of the space.

- One descriptive feature $\rightarrow$ hyperplane is a point (of dimension $0 = n-1$)
- Two descriptive features $\rightarrow$ hyperplane is a line (of dimension $1 = n-1$)
- Three descriptive features $\rightarrow$ hyperplane is a plane (of dimension $2 = n-1$)
- ...

For $n > 3$, the problem can't be visualized anymore, but SVMs also work for very large n.

- This is in contrast to logistic regression, which gets too complex at large n

Hyperplane

The technical definition of the **hyperplane** is as follows:

$$\vec{\mathbf{w}} \cdot \vec{\mathbf{x}} + b = \underbrace{\mathbf{w}_1\mathbf{x}_1 + \mathbf{w}_2\mathbf{x}_2 + \cdots + \mathbf{w}_n\mathbf{x}_n}_{\substack{\vec{\mathbf{x}}=(x_1,x_2,\ldots,x_n) \\ \vec{\mathbf{w}}=(w_1,w_2,\ldots,w_n)}} + b = 0$$

At least one of the weights is non-zero. The hyperplane then splits the n-dimensional into two disjoint parts:

- $\vec{\mathbf{w}} \cdot \vec{\mathbf{x}} + b \geq 0$
- $\vec{\mathbf{w}} \cdot \vec{\mathbf{x}} + b < 0$

There are some cases where the dataset is not separable.

- This can be due to **outliers**. A solution for this problem is to relax the problem by allowing violating instances $\rightarrow$ but they then add a penalty.
- Alternatively, there can also be a **structural** problem (not linearly separable in n-dimensional space). The idea to solve this problem is to try whether the problem is linearly separable in a higher-dimensional space, which then also includes feature design.

We need to make some further definitions to completely formalize the SVM approach. We'll first look at **vectors** like $\vec{\mathbf{x}} = (x_1, x_2, \ldots, x_n)$. Vector

- The vector **length** is defined as Vector length

$$\| \vec{\mathbf{x}} \| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

- The vector **direction** is the normalized vector of length one, so Vector direction

$$\vec{\mathbf{w}} = \frac{\vec{\mathbf{x}}}{\| \vec{\mathbf{x}} \|} = \left(\frac{x_1}{\| \vec{\mathbf{x}} \|}, \frac{x_2}{\| \vec{\mathbf{x}} \|}, \cdots, \frac{x_n}{\| \vec{\mathbf{x}} \|} \right)$$

- To extend the length of a vector without changing its direction, multiply the vector with a constant

$$q\vec{\mathbf{x}} = (q \cdot x_1, q \cdot x_2, \ldots, q \cdot x_n)$$

- The dot product on the other hand allows the multiplication of two vectors:

$$\vec{\mathbf{x}} \cdot \vec{\mathbf{y}} = \sum_{i=1}^{n} x_i y_i = \underbrace{\| \vec{\mathbf{x}} \| \cdot \| \vec{\mathbf{y}} \| \cdot \cos\theta}_{\text{with angle } \theta \text{ between } x,y} \quad \in \mathbb{R}$$

  - The dot product is heavily influenced by the angle between the two vectors and therefore also tells their linear dependency.
  - For orthogonal vectors $\cos\theta = 0$, for vectors with the same direction $\cos\theta = 1$ and for angles in the opposite direction $\cos\theta = -1$

Now, let's look deeper into the definition of a hyperplane. Our hyperplane is defined by

$$\vec{\mathbf{w}} \cdot \vec{\mathbf{x}} + b = 0 \iff \sum_{i=1}^{n} w_i x_i + b = 0$$

where

- $\vec{\mathbf{x}} = (x_1, x_2, \ldots, x_n)$ are free variables,
- $\vec{\mathbf{w}} = (w_1, w_2, \ldots, w_n)$ is the direction vector of the hyperplane (no-zero),
- And the hyperplane is fully defined by $\vec{\mathbf{w}}$ and b

Further interesting to see:

$$\text{For } q \in \mathbb{R} \setminus \{0\}: \quad \vec{\mathbf{w}} \cdot \vec{\mathbf{x}} + b = 0 \iff \sum_{i=1}^{n} w_i x_i + b = 0$$
$$\iff \quad q\vec{\mathbf{w}} \cdot \vec{\mathbf{x}} + qb = 0 \iff \sum_{i=1}^{n} q w_i x_i + qb = 0$$

The hyperplane splits a set of data points in the following way. Assume we have N data points $X_N = \{\vec{\mathbf{x}}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,n}) \,|\, i \in N\}$. Then, for $\vec{\mathbf{x}}_i$ calculate $f(\vec{\mathbf{x}}_i) := \vec{\mathbf{w}} \cdot \vec{\mathbf{x}}_i + b = \sum_{j=1}^{n} w_j x_{i,j} + b$, and

$$\text{Classifiy as } \begin{matrix}\text{positive} \\ \text{negative}\end{matrix} \text{ if } f(\vec{\mathbf{x}}_i) \begin{matrix}\geq 0 \\ < 0\end{matrix}$$

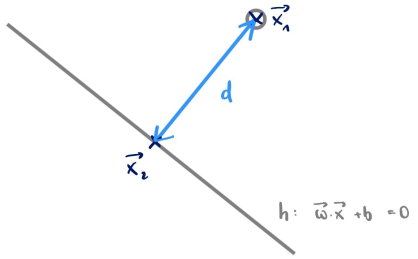
Note the ongoing confusion of whether above or below the hyperplane is defined as positive, e.g. by multiplying $\vec{\mathbf{w}}$ and b with $q = -1$

Perfectly separating hyperplane

By splitting the data as described above, we introduce two possible classes for each $\vec{\mathbf{x}}_i$, namely $y_i \in \{-1, 1\}$. If our hyperplane $(\vec{\mathbf{w}}, b)$ **separates** a dataset X_N **perfectly**, then:

$$\text{For all instances } (\vec{\mathbf{x}}_i, y_i): \quad \begin{matrix} y_i = +1 & \implies & f(\vec{\mathbf{x}}_i) \geq 0 \\ y_i = -1 & \implies & f(\vec{\mathbf{x}}_i) < 0 \end{matrix}$$

But actually, there might be many "perfect" hyperplanes. The one we would like to pick is the one that maximizes the distance of the hyperplane to the nearest point. For that, we of course first need to define the **distance between a point and a hyperplane**. Assume, we have the following setting:



Assume any point $\vec{\mathbf{x}}_1$ and hyperplane $(\vec{\mathbf{w}}, b)$

- $\vec{\mathbf{x}}_2$ is closest point on the hyperplane
- d as distance between $\vec{\mathbf{x}}_1$ and the hyperplane, defined as the distance between the two points

Figure 1.3: Distance between a point and the hyperplane

As can be seen in figure 1.3, we have the closest point defined as the point that can be found by following an orthogonal path and seeing which point is on the hyperplane:

$$\underbrace{\vec{\mathbf{x}}_1 = \vec{\mathbf{x}}_2 + \lambda\vec{\mathbf{w}}}_{\text{orthogonal path}} \text{ and } \underbrace{\vec{\mathbf{w}} \cdot \vec{\mathbf{x}}_2 + b = 0}_{\vec{\mathbf{x}}_2 \text{ is on hyperplane}}$$
$$\implies \quad \vec{\mathbf{w}} \cdot (\vec{\mathbf{x}}_1 - \lambda\vec{\mathbf{w}}) + b = 0$$
$$\implies \quad d = \|\lambda\vec{\mathbf{w}}\|$$

Next, we try to get rid of the λ in the computation of $d = \|\lambda\vec{\mathbf{w}}\|$. For that, just rewrite

the formula for the calculation of $\vec{x}_2$:

$$\begin{aligned}
& \vec{w} \cdot (\vec{x}_1 - \lambda \vec{w}) + b = 0 \\
\Longrightarrow \quad & \vec{w} \cdot \vec{x}_1 + b = \lambda \vec{w} \cdot \vec{w} = \lambda \|\vec{w}\|^2 \\
\Longrightarrow \quad & \lambda = \frac{\vec{w} \cdot \vec{x}_1 + b}{\|\vec{w}\|^2} \\
\Longrightarrow \quad & d = \left\| \left(\frac{\vec{w} \cdot \vec{x}_1 + b}{\|\vec{w}\|^2} \right) \cdot \vec{w} \right\| \\
& = \left| \frac{\vec{w} \cdot \vec{x}_1 + b}{\|\vec{w}\|^2} \right| \|\vec{w}\| = \left| \frac{\vec{w} \cdot \vec{x}_1 + b}{\|\vec{w}\|} \right|
\end{aligned}$$

So finally, we can formulate how to calculate the **optimal hyperplane**: Optimal hyperplane

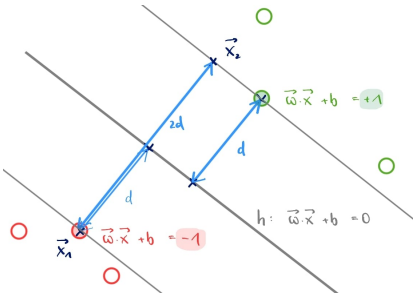
Given $\left\{ (\vec{x}_i, y_i) \in \mathbb{R}^n \times \{-1, 1\} \,\middle|\, 1 \leq i \leq m \right\}$

Find $(\vec{w}, b)$ such that for any $1 \leq i \leq m$: If $y_i = +1$, then $\vec{w} \cdot \vec{x}_i + b \geq 0$; If $y_i = -1$, then $\vec{w} \cdot \vec{x}_i + b < 0$

And maximize $\min_{1 \leq i < m} d_i$ with : $d_i = \left| \frac{\vec{w} \cdot \vec{x}_i + b}{\|\vec{w}\|} \right|$

Alternatively, we could normalize the distance to one, such that the resulting situation is as depicted in 1.4.

- This rescaling can be performed since the hyperplane formulation doesn't change when multiplying with a constant q.
- The support vectors yield the result $\vec{w} \cdot \vec{x}_i + b = \pm 1$.



The support vectors are on the lines
(positive and negative support vectors)

Figure 1.4: Normalized distance

Also, the classification is adjusted to

$$\left.\begin{array}{l} \text{If } y_i = +1 \quad \text{, then } \vec{w} \cdot \vec{x}_i + b \geq +1 \\ \text{If } y_i = -1 \quad \text{, then } \vec{w} \cdot \vec{x}_i + b \leq -1 \end{array}\right\} \rightarrow \quad \begin{array}{l} \text{can be combined to:} \\ \forall i : y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 \end{array}$$

Optimal hyperplane normalized

Our goal is still to maximize the smallest $d = \|\lambda \vec{w}\|$. In our alternative formulation, we want to formalize this goal only with the help of the shifted hyperplanes $\vec{x} \cdot \vec{x} + b = \pm 1$:

- Choose an arbitrary support vector $\vec{x}_1$ (let's say on $\vec{x} \cdot \vec{x} + b = -1$)
- Now take the corresponding $\vec{x}_2 = \vec{x}_1 + 2\lambda \vec{w}$ on the other support-vector-line (so $\vec{w} \cdot \vec{x} + b = +1$)

- This leads to the following equations to be solved:

$$\vec{\mathbf{w}} \cdot \vec{\mathbf{x}}_1 + b = -1$$
$$\vec{\mathbf{w}} \cdot (\vec{\mathbf{x}}_1 + 2\lambda\vec{\mathbf{w}}) + b = +1$$

- This can be further readjusted to:

$$\implies \quad \underbrace{\vec{\mathbf{w}} \cdot \vec{\mathbf{x}}_1 + b}_{=-1} + \vec{\mathbf{w}} \cdot 2\lambda\vec{\mathbf{w}} = +1$$
$$\implies \quad \vec{\mathbf{w}} \cdot 2\lambda\vec{\mathbf{w}} = +2$$
$$\implies \quad \lambda = \frac{1}{\vec{\mathbf{w}} \cdot \vec{\mathbf{w}}} = \frac{1}{\|\vec{\mathbf{w}}\|^2}$$

- So, we can replace:

$$\text{Maximize } d = \|\lambda\vec{\mathbf{w}}\| = \frac{1}{\|\vec{\mathbf{w}}\|^2}\|\vec{\mathbf{w}}\| = \frac{1}{\|\vec{\mathbf{w}}\|}$$
$$\text{Minimize } \|\vec{\mathbf{w}}\| \text{ under the constraints stated before}$$

So, the reformulated problem is:

Reformulated: SVM problem

$$\text{Given a set of } m \text{ instances } \left\{(\vec{\mathbf{x}}_i, y_i) \in \mathbb{R}^n \times \{-1, 1\} \,\middle|\, 1 \leq i \leq m\right\}$$
$$\min_{\vec{\mathbf{w}}, b} \|\vec{\mathbf{w}}\| \text{ s.t. } \forall i : y_i(\vec{\mathbf{w}} \cdot \vec{\mathbf{x}}_i + b) \geq 1$$

For technical reasons, a common variant of this problem is the convex quadratic optimization problem

Convex quadratic optimization problem

$$\text{Given a set of } m \text{ instances } \left\{(\vec{\mathbf{x}}_i, y_i) \in \mathbb{R}^n \times \{-1, 1\} \,\middle|\, 1 \leq i \leq m\right\}$$
$$\min_{\vec{\mathbf{w}}, b} \frac{1}{2}\|\vec{\mathbf{w}}\|^2 \text{ s.t. } \forall i : y_i(\vec{\mathbf{w}} \cdot \vec{\mathbf{x}}_i + b) \geq 1$$

- This problem is easier to solve than the original linear formulation (maximizing $\min_{1 \leq i < m} d_i$)
- The concrete solution or techniques for finding solutions are not covered here, but assume they exist

Another reformulation of the problem is the Wolfe dual Lagrangian problem which is important for the "kernel trick":

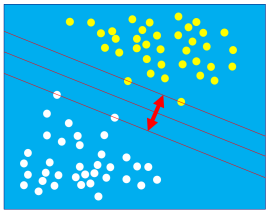
Wolfe dual Lagrangian problem

$$\text{For } \vec{\mathbf{w}} = \sum_{i=1}^{m} \alpha_i y_i \vec{\mathbf{x}}_i \text{ and } \alpha_i \geq 0, i = 1, \ldots, m$$
$$\begin{aligned} \max_{\alpha} \quad & \textstyle\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \vec{\mathbf{x}}_i \vec{\mathbf{x}}_j \\ \text{subject to} \quad & \textstyle\sum_{i=1}^{m} \alpha_i y_i = 0 \end{aligned}$$

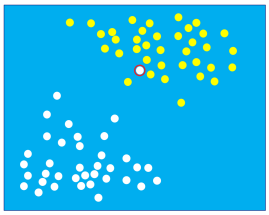
1.2 Soft-margin SVMs

So far, we looked at the general problem formulation of SVMs when the dataset was separable. Next, we're gonna look at how to handle datasets where no reasonable hyperplane can be found to separate the instances, even after removing outliers.

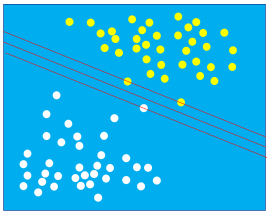
To see, what the problem is, consider the examples from figure 1.5.



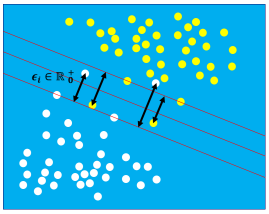
(a) Nicely separable data



(b) One outlier destroying separability



(c) Outlier reducing margin



(d) Reasonable hyperplane tolerating outliers
$\rightarrow$ add goal of minimizing slack variables

Figure 1.5: Separability of datasets

So, the reformulated problem is the following:

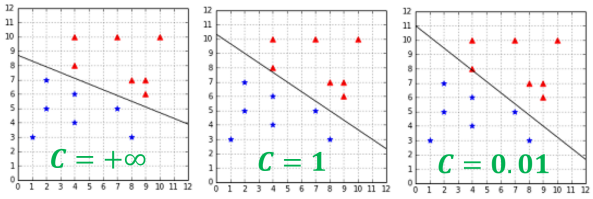
Given a set of m instances $\left\{(\vec{\mathbf{x}}_i, y_i) \in \mathbb{R}^n \times \{-1, 1\} \,\middle|\, 1 \leq i \leq m\right\}$

Soft-margin SVM problem

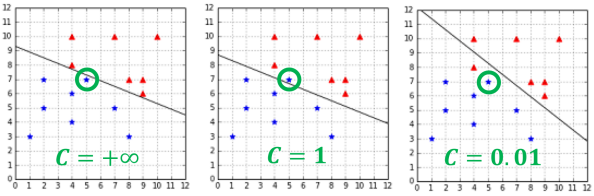
$$\min_{\vec{\mathbf{w}}, b, \epsilon} \frac{1}{2}\|\vec{\mathbf{w}}\|^2 + C\sum_{i=1}^{m} \epsilon_i \text{ s.t. } \forall i : y_i(\vec{\mathbf{w}} \cdot \vec{\mathbf{x}}_i + b) \geq 1 - \epsilon_i \text{ where } \epsilon_i \geq 0$$

The role of C determines how much we want to restrict tolerance for outliers.

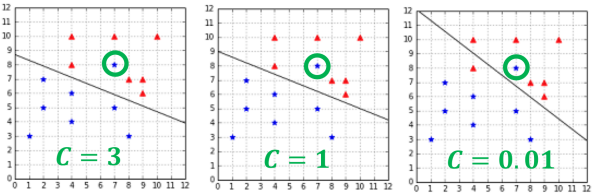
- For a high C, we have a strict formulation of our problem, so no outlier tolerance.
- For a low C, we have a more flexible formulation of the problem, so outliers are tolerated.
- Consider 1.6 to see the effect of C on different datasets.



(a) Separable dataset



(b) One outlier, but still separable dataset



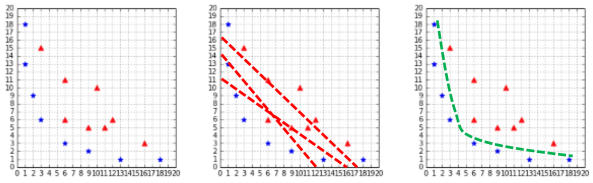
(c) Not separable dataset
Especially for $C = +\infty$: no solution of SVM exists for this problem

Figure 1.6: Role of C for soft-margin SVM

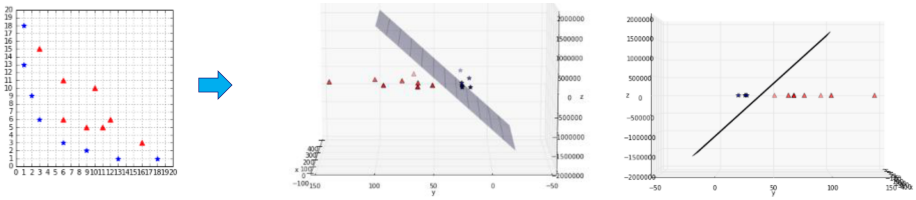
1.3 Non-linear decision boundaries

Next, we're considering non-linear separation, where our dataset can't be separated by a linear hyperplane, but by some other non-linear decision boundary. Consider the example in 1.7. Our idea to solve this separation is to **lift the number of dimensions**.

Lift number of dimensions



(a) Non-linear decision boundary



(b) Lift from $n = 2$ to $n = 3$

Figure 1.7: Non-linear separation

The mapping to lift the dimensions is similar to the one introduced for regression. So, we have the updated problem formulation:

Soft-margin, dimension-lifted SVM problem

$$\text{Given a set of } m \text{ instances } \left\{(\vec{\mathbf{x}}_i, y_i) \in \mathbb{R}^n \times \{-1, 1\} \,\middle|\, 1 \leq i \leq m\right\}$$
$$\text{Find a mapping } \phi \in \mathbb{R}^n \to \mathbb{R}^q \text{ with } q > n$$
$$\text{E.g.: } \phi_0(x) = 1, \phi_1(x) = x, \phi_2(x) = x^2, \ldots$$
$$\min_{\vec{\mathbf{w}}, b, \epsilon} \frac{1}{2}\|\vec{\mathbf{w}}\|^2 + C\sum_{i=1}^{m} \epsilon_i \text{ s.t. } \forall i : y_i(\vec{\mathbf{w}} \cdot \phi(\vec{\mathbf{x}}_i) + b) \geq 1 - \epsilon_i \text{ where } \epsilon_i \geq 0$$

For our example 1.7, the mapping function was formulated as:

$$\phi \in \mathbb{R}^2 \to \mathbb{R}^3$$
$$\phi(x_1, x_2) = \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right)$$

1.4 Using kernels

Let's look again at the **Wolfe dual Lagrangian problem**, but this time applied to the dimension-lifting trick from before.

Wolfe dual Lagrangian problem with kernel

$$\text{For } \vec{\mathbf{w}} = \sum_{i=1}^{m} \alpha_i y_i \vec{\mathbf{x}}_i \text{ and } \alpha_i \geq 0, i = 1, \ldots, m$$
$$\max_\alpha \quad \sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \underbrace{K(\vec{\mathbf{x}}_i, \vec{\mathbf{x}}_j)}_{=\phi(\vec{\mathbf{x}}_i)\cdot\phi(\vec{\mathbf{x}}_j)}$$
$$\text{subject to} \quad \sum_{i=1}^m \alpha_i y_i = 0$$

- Once again, we have $\phi \in \mathbb{R}^n \to \mathbb{R}^q$ with $q > n$
- K is our **kernel function** — Kernel function
  - It efficiently computes $\phi(\vec{\mathbf{x}}_i) \cdot \phi(\vec{\mathbf{x}}_j)$
  - S.t., the actual mapping mappings $\phi(\vec{\mathbf{x}}_i)$ and $\phi(\vec{\mathbf{x}}_j)$ as well as their dot product don't need to be computed

Consider the kernel function for our previous example 1.7.

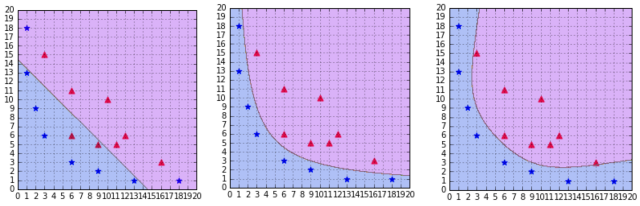
$$\begin{aligned}
\phi(x_1, x_2) &= \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right) \\
\implies \phi(x_{i/j,1}, x_{i/j2}) &= \left(x_{i/j,1}^2, \sqrt{2}x_{i/j,1}x_{i/j,2}, x_{i/j,2}^2\right) \\
\implies K(\vec{\mathbf{x}}_i, \vec{\mathbf{x}}_j) = \phi(\vec{\mathbf{x}}_i) \cdot \phi(\vec{\mathbf{x}}_j) &= \underbrace{x_{i,1}^2 x_{j,1}^2}_{(x_{i,1}x_{j,1})^2} + \underbrace{2x_{i,1}x_{i,2}x_{j,1}x_{j,2}}_{2(x_{i,1}x_{j,1})(x_{i,2}x_{j,2})} + \underbrace{x_{i,2}^2 x_{j,2}^2}_{(x_{i,2}x_{j,2})^2} \\
&= (x_{i,1}x_{j,1} + x_{i,2}x_{j,2})^2
\end{aligned}$$

- So, for calculating $\phi(\vec{\mathbf{x}}_i)\cdot\phi(\vec{\mathbf{x}}_j)$, we need 10 multiplication-operations and 2 addition-operations,
- Whereas for calculating $K(\vec{\mathbf{x}}_i, \vec{\mathbf{x}}_j)$, we need 3 multiplication-operations and 1 addition-operation.

In the following, we'll see some kernel functions:

- We can use **polynomial kernels** with parameters c (constant) and d (degree, remember the danger of overfitting with increasing degree).

$$K(\vec{\mathbf{x}}_i, \vec{\mathbf{x}}_j) = \underbrace{\phi(\vec{\mathbf{x}}_i) \cdot \phi(\vec{\mathbf{x}}_j)}_{\phi \text{ may have a variable (large) number of dimensions}} = (\vec{\mathbf{x}}_i \cdot \vec{\mathbf{x}}_j + c)^d$$



$d = 1$ $d = 2$ $d = 6$

- Alternatively, there are e.g. **circular kernels**, since some data can't be separated by any polynomial.