Handout

# Introduction to Machine Learning

Kernel Methods

*Leonie Pätzold*

supervised by PhD. Akash Ashirbad Panda

18.01.2021

# Contents

**1 Introduction** **3**

**2 The Kernel Trick: Dual Representations** **3**

**3 Constructing Kernels** **5**

**4 Examples** **7**

4.1 Radial Basis Function Networks: The Nadaraya-Watson Model . . . . . 7

4.2 Polynomial Kernels in Classification . . . . . . . . . . . . . . . . . . . . . . . 9

4.3 Graph kernels . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

# 1 Introduction

In the previous talks we learned about supervised and unsupervised learning, linear regression and classification.
Today we focus on the prediction function, which was introduced in the talk about linear models of regression by Nadia Vohwinkel. We will learn how to express the prediction function by using kernels, which you can think of as a scalar product of some sort. This means instead of training the weights by using the training data, we use the training data in the kernels.
Remember for instance the method of nearest neighbour, which is used for classification problems. Here some sort of metric is needed to define closeness of points and clusters. In this memory based method, all or at least most training points need to remembered to define the clusters and to predict the right cluster affiliation for a new input.
Very common kernel functions are for instance the *linear kernel* $k(x,x') = x^T x'$, *stationary kernels* $k(x,x') = k(x-x')$ and *homogeneous kernels* $k(x,x') = k(||x-x'||)$, which are also known as *radial basis functions.*

We now design the prediction function in such a way that we use the input data $x$ only in the kernels. This way we can profit from the advantages a kernel brings. In particular we can use infinite feature spaces and are not limited to strictly numerical inputs x. We can for instance have sets or graphs as input.

# 2 The Kernel Trick: Dual Representations

In this chapter we want to explore the possibilities to express prediction functions, we have already encountered, via kernels. In particular we have encountered linear parametric models in the talks about linear models for regression and classification. There we used the weight $\mathbf{w} = (w_1, \cdots, w_M)$ and basis functions $\phi_i(x)$, with the feature space mapping $\phi(x) = (\phi_1, \cdots, \phi_M)(x)$ to express the prediction function

$$y(\mathbf{x}) = \sum_{i=1}^{M} w_i \phi_i(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}). \tag{1}$$

We now want to use the kernel

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}') \tag{2}$$

in the prediction function without using the weights. Instead of the weights we will use the training data $\mathbf{x}_1, \cdots, \mathbf{x}_n$.
Let us remember the example in chapter 1.2.5 on the handout of Linear Models of regression by Nadia Vohwinkel. There we had a linear regression model, where the parameter $\mathbf{w}$ is determined by minimizing a regularized sum-of-squares.
The regularized sum-of-squares error function was given by

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} (\mathbf{w}^{\mathbf{T}} \phi(\mathbf{x}_n) - t_n)^2 + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}, \tag{3}$$

with $\lambda > 0$ and $\phi = (\phi_1, ..., \phi_{M-1})$, where the $\phi_i$ are the basis functions and $\mathbf{w}$ the weight vector.
We now need to express $\mathbf{w}$ with the basis functions. This we do by setting the gradient of $E(\mathbf{w})$ to 0.
The solution for $\mathbf{w}$ takes the form of a linear combination of the vectors $\phi(x_n)$, with coefficients $a_n$ that are functions of $\mathbf{w}$, of the form

$$\mathbf{w} = -\frac{1}{\lambda}\sum_{n=1}^{N}\{\mathbf{w}^T\phi(\mathbf{x}_n) - t_n\}\phi(\mathbf{x}_n) = \sum_{n=1}^{N} a_n\phi(\mathbf{x}_n) = \mathbf{\Phi}^T\mathbf{a} \tag{4}$$

where $\mathbf{\Phi}$ is the design matrix, whose $n^{th}$ row is given by $\phi(\mathbf{x}_n)^T$ and the vector $\mathbf{a} = (a_1, \cdots, a_N)$ is defined by

$$a_n = -\frac{1}{\lambda}\{\mathbf{w}^T\phi(\mathbf{x}_n) - t_n\}. \tag{5}$$

We can now reformulate the least squares algorithm in terms of the parameter vector $\mathbf{a}$, giving rise to a *dual representation.* We can now substitute $\mathbf{w} = \mathbf{\Phi}^T\mathbf{a}$ and get

$$E(\mathbf{a}) = \frac{1}{2}\mathbf{a}^T\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{a} - \mathbf{a}^T\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^T\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{a}$$

where $\mathbf{t} = (t_1 \cdots, t_N)^T$. We can now define the *Gram* matrix $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^T$, which is an $N \times N$ symmetric matrix with elements

$$K_{nm} = \phi(\mathbf{x}_n)^T\phi(\mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m).$$

We can now see that we use the kernel (2) here. This means we are half way, where we want to be. Using the Gram matrix in the sum-of-squares error function gives us

$$E(\mathbf{a}) = \frac{1}{2}\mathbf{a}^T\mathbf{K}\mathbf{K}\mathbf{a} - \mathbf{a}^T\mathbf{K}\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^T\mathbf{K}\mathbf{a}.$$

If we now set the gradient of $E(\mathbf{a})$ with respect to $\mathbf{a}$ to zero, we get

$$\mathbf{a} = (\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\mathbf{t}.$$

Substitution back into (1) gives us the following prediction function

$$y(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) = \mathbf{a}^T\mathbf{\Phi}\phi(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T(\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\mathbf{t}$$

where we have defined the vector $\mathbf{k}(\mathbf{x}) = (k_1(\mathbf{x}), \cdots, k_N(\mathbf{x}))$ with elements $k_i(\mathbf{x}) = k(\mathbf{x}_i, \mathbf{x})$.

To recap, we see that in the dual formulation, we determine the parameter vector $\mathbf{a}$ by inverting an $N \times N$ matrix, whereas in the original parameter space we had to invert an $M \times M$ matrix in order to determine $\mathbf{w}$.
As $N$ is typically much larger than $M$ the dual formulation does not seem very useful. However in the dual representation we express the data entirely in the form of the kernel function $k(\mathbf{x}, \mathbf{x}')$, so we can work directly with the kernels and do not have to worry about the size of the feature space. In particular this offers the opportunity

to work with feature spaces with high or even infinite dimensionality. This is a huge advantage in the use of kernels.

## 3 Constructing Kernels

In this chapter we focus on the way to construct valid kernels and first of all we will give a formal definition for a valid kernel, which was taken from [PV, S.4].

**Definition 3.1.** A function $k : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$ is called a *valid* or *positive definite kernel* iff it is symmetric, that is, $k(x, x') = k(x', x)$ for all $x, x' \in \mathcal{X}$, and positive definite, that is,

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(x_i, x_j) \geq 0$$

for any $n > 0$, any choice of n objects $x_1, \cdots, x_n \in \mathcal{X}$, and any choice of real numbers $c_1, \cdots, c_n \in \mathbb{R}$.

There are different methods to construct valid kernels. One way to do it is to choose a feature space mapping $\phi(x)$ and then find the corresponding kernel

$$k(x, x') = \phi(x)^T \phi(x') = \sum_{i=1}^{M} \phi_i(x)\phi_i(x') \tag{6}$$

where $\phi_i(x)$ are the basis functions.
This kernel is a valid kernel as it is obviously symmetric and we can see that for any $n > 0$, $x_1, \cdots, x_n \in \mathcal{X}$ and $c_1, \cdots c_n \in \mathbb{R}$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(x_i, x_j) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j \phi(x_i)^T \phi(x_j') = ||\sum_{i=1}^{n} c_i \phi(x_i)||^2 \geq 0 \tag{7}$$

holds. Note that we get the valid kernel $k(\mathbf{x}, \mathbf{x})' = \mathbf{x}^T \mathbf{x}'$ if we take the basis functions $\phi_i(\mathbf{x}) = x_i$ for all $i = 1, \ldots, N$.

Another method is to try to construct a kernel function directly. In this case we have to prove, that the constructed kernel is a valid kernel. As a simple example, we consider a kernel function given by

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z})^2. \tag{8}$$

If we assume we have a two dimensional input space $\mathbf{x} = (x_1, x_2)$ we can expand the terms to find the corresponding basis functions

$$\begin{aligned} k(\mathbf{x}, \mathbf{z}) &= (\mathbf{x}^T \mathbf{z})^2 = (x_1 z_1 + x_2 z_2)^2 \\ &= x_1^2 z_1^2 + 2 x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\ &= (x_1^2, \sqrt{2} x_1 x_2, x_2^2)(z_1^2, \sqrt{2} z_1 z_2, z_2^2) \\ &= \phi(\mathbf{x})^T \phi(\mathbf{z}). \end{aligned} \tag{9}$$

Here we see, that we have three basis functions $\phi_1(x) = x_1^2$, $\phi_2(x) = \sqrt{2}x_2x_2$ and $\phi_3(x) = x_2^2$.
However, in practise constructing the feature mapping with its basis functions is not always that easy. Therefore a different, but quite powerful technique for constructing new kernels is used.
The idea is to build them out of simpler kernels as building blocks. This can be done using the following properties.

**Lemma 3.2.** *Given valid kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$ the following kernels will also be valid:*

$$k(\mathbf{x}, \mathbf{x}') = ck_1(\mathbf{x}, \mathbf{x}') \tag{10}$$
$$k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})k_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}') \tag{11}$$
$$k(\mathbf{x}, \mathbf{x}') = q(k_1(\mathbf{x}, \mathbf{x}')) \tag{12}$$
$$k(\mathbf{x}, \mathbf{x}') = \exp(k_1(\mathbf{x}, \mathbf{x}')) \tag{13}$$
$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \tag{14}$$
$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') \cdot k_2(\mathbf{x}, \mathbf{x}') \tag{15}$$
$$k(\mathbf{x}, \mathbf{x}') = k_3(\phi(\mathbf{x}), \phi(\mathbf{x}')) \tag{16}$$
$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{A}\mathbf{x}' \tag{17}$$
$$k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b) \tag{18}$$
$$k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}'_a) \cdot k_b(\mathbf{x}_b, \mathbf{x}'_b) \tag{19}$$

*where $c > 0$ is a constant, $f(\cdot)$ is any function and $q(\cdot)$ is a polynomial with non negative coefficients, $\phi(\mathbf{x})$ is a function from $\mathbf{x}$ to $\mathbb{R}^M$, $k_3(\cdot, \cdot)$ is a valid kernel in $\mathbb{R}^M$, $\mathbf{A}$ is a symmetric positive semi definite matrix, $\mathbf{x}_a$ an $\mathbf{x}_b$ are variables (not necessarily disjoint) with $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$, and $k_a$ and $k_b$ are valid kernel functions over their respective spaces.*

Equipped with these properties, we will now construct complex kernels.

**Proposition 3.3.** *The kernel $k_1(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}' + c)^2$ with $c > 0$, $k_2(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}')^M$ with $M \in \mathbb{N}$ and $k_3(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}' + c)^M$ with $c > 0$ and $M \in \mathbb{N}$ are valid kernels.*

*Proof.* As we know from above $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}')^2$ is a simple polynomial kernel. In particular it is a valid kernel. We only showed it for a two dimensional input space, but it is true for all. As c is positive we can define $q(x) = (x + c)^2$ and by applying (12) from lemma 3.2 we see that the kernel $k_1(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}' + c)^2$ with $c > 0$ is valid. As we know $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{x}'$ is a valid kernel, we can apply (12) $M$ times and get a valid kernel $k_2$.
Analog to the first two cases we can use (12) show that $k_3$ is a valid kernel. □

**Remark 3.4.** Imagine $\mathbf{x}$ and $\mathbf{x}'$ are two images. Then the kernel $k_2$ represents a particular weighted sum of all possible products of $M$ pixels in the first image with $M$ pixels in the second image. This can be generalised to include all terms up to degree $M$ by using kernel $k_3$.

**Proposition 3.5.** *The 'Gaussian' kernel*

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\sigma^2}) \tag{20}$$

*is valid.*

*Proof.* We can see that this kernel is valid by first expanding the squares

$$||\mathbf{x} - \mathbf{x}'||^2 = \mathbf{x}^T\mathbf{x} + \mathbf{x}'^T\mathbf{x}' - 2\mathbf{x}^T\mathbf{x}' \tag{21}$$

to get

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\frac{\mathbf{x}^T\mathbf{x}}{2\sigma^2})\exp(-\frac{(\mathbf{x}')^T\mathbf{x}'}{2\sigma^2})\exp(\frac{\mathbf{x}^T\mathbf{x}'}{\sigma^2}). \tag{22}$$

Now we can use (11) and (13) together with the validity of the linear kernel $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{x}'$. □

**Remark 3.6.** The Gaussian kernel is not restricted to the use of the Euclidean distance. We can for instance use a kernel substitution in (21) to replace $\mathbf{x}^T\mathbf{x}$ with a non linear kernel $\kappa(\mathbf{x}, \mathbf{x}')$

$$k(\mathbf{x}, \mathbf{x}') = \exp(-(\kappa(\mathbf{x}, \mathbf{x}) + \kappa(\mathbf{x}', \mathbf{x}') - 2\kappa(\mathbf{x}, \mathbf{x}'))/2\sigma^2). \tag{23}$$

## 4 Examples

### 4.1 Radial Basis Function Networks: The Nadaraya-Watson Model

In the talk about linear regression models by Nadia Vohwinkel we encountered basis functions, but did not specify their form. One choice is as radial basis functions, which have the property that each basis function depends only on the radial distance from a center $\mu_j$, so that for $M \in \mathbb{N}$ we have $\phi_j(x) = h(||x - \mu_j||)$ for all $j = 1, \ldots, M$.
Originally radial basis functions were used for exact function interpolation. So given input vectors $\mathbf{x}_1, \cdots, \mathbf{x}_N$ and their target values $t_1, \cdots, t_N$ the aim was to find a function $f$ such that $f(x_i) = t_i$ for all $i = 1 \cdots, N$. This can be archived by a linear combination of radial basis functions

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i h(||\mathbf{x} - \mathbf{x}_i||)$$

where the values of the coefficients $w_i$ are found by least squares.
As we have the same amount of coefficients as constraints, the resulting function fits every target value exactly. But as we discussed in previous talks before this is not ideal in machine learning. As the training data, consisting of the input values and the target values, normally has some sort of noise, this technique can easily result in an over-fitted solution, which is not desirable.
We may assume now that the input variables are noisy. Furthermore, the noise can be described by a variable $\boldsymbol{\xi}$ with the distribution $\nu(\boldsymbol{\xi})$. This way we get the following sum-of-squares error function

$$E = \frac{1}{2}\sum_{i=1}^{N} \int (y(\mathbf{x}_i + \boldsymbol{\xi}) - t_i)^2 \nu(\boldsymbol{\xi}) d\boldsymbol{\xi}. \tag{24}$$

We can now optimize with respect to the function $f(\mathbf{x})$ and get

$$y(\mathbf{x}) = \sum_{i=1}^{N} t_i h(\mathbf{x} - \mathbf{x}_i) \tag{25}$$

where the basis functions are given by

$$\phi_i(\mathbf{x}) = h(\mathbf{x} - \mathbf{x}_i) = \frac{\nu(\mathbf{x} - \mathbf{x}_i)}{\sum\limits_{n=1}^{N} \nu(\mathbf{x} - \mathbf{x}_n)}. \tag{26}$$

Each basis function is centered on one data point. This is known as the *Nadaraya-Watson* model. If the noise distribution $\nu(\boldsymbol{\xi})$ is a function only of $||\boldsymbol{\xi}||$, then the basis functions will be radial, so that

$$\phi_i(\mathbf{x}) = h(||\mathbf{x} - \mathbf{x}_i||) = \frac{\nu(||\mathbf{x} - \mathbf{x}_i||)}{\sum\limits_{n=1}^{N} \nu(||\mathbf{x} - \mathbf{x}_n||)}. \tag{27}$$

We should note here that the basis functions are normalised, so that $\sum_i h(\mathbf{x} - \mathbf{x}_i) = 1$ for any value of $\mathbf{x}$. This normalisation ensures that one avoids having regions in the input space where all of the basis functions have a small value.

We will now have a look at the following: Suppose we have a training data set $\mathbf{x}_1, \cdots, \mathbf{x}_N$ with corresponding target values $t_1, \cdots, t_N$ and we model the joint distribution $p(\mathbf{x}, t)$ so that

$$p(\mathbf{x}, t) = \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{x} - \mathbf{x}_i, t - t_i), \tag{28}$$

where $f(\mathbf{x}, t)$ is the component density function. We can now express the prediction function $y(\mathbf{x})$ the following way

$$\begin{aligned} y(\mathbf{x}) = \mathbb{E}[t|\mathbf{x}] &= \int_{-\infty}^{\infty} tp(t|\mathbf{x})dt \\ &= \frac{\int tp(t|\mathbf{x})dt}{\int p(t|\mathbf{x})dt} \\ &= \frac{\sum_i \int tf(\mathbf{x} - \mathbf{x}_i, t - t_i)dt}{\sum_n \int f(\mathbf{x} - \mathbf{x}_n, t - t_n)dt}. \end{aligned}$$

We assume for simplicity that the integral over $f(\mathbf{x}, t)t$ is zero for all values of $\mathbf{x}$ and using a change of variables namely $g(x) := \int f(\mathbf{x}, t)dt$, we get

$$y(\mathbf{x}) = \frac{\sum_i g(\mathbf{x} - \mathbf{x}_i, t)t_i}{\sum_n g(\mathbf{x} - \mathbf{x}_n, t)} = \sum_i k(x, x_i)t_i \tag{29}$$

where $i, n = 1, \ldots, N$ and the kernel function is given by

$$k(\mathbf{x}, \mathbf{x}_i) = \frac{\sum_i g(\mathbf{x} - \mathbf{x}_i, t)}{\sum_n g(\mathbf{x} - \mathbf{x}_n, t)}. \tag{30}$$

An example for a Nadaraya Watson kernel regression model with isotropic Gaussian kernels is given in Figure 0.

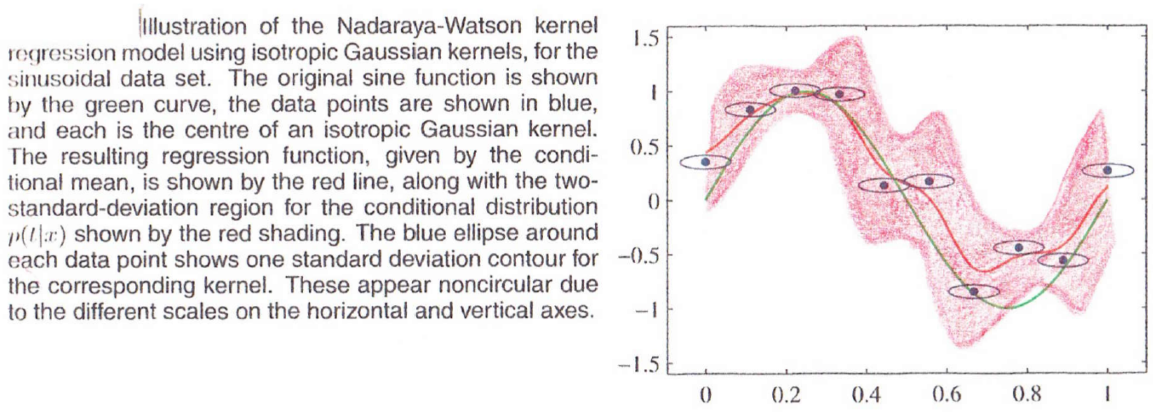

Illustration of the Nadaraya-Watson kernel regression model using isotropic Gaussian kernels, for the sinusoidal data set. The original sine function is shown by the green curve, the data points are shown in blue, and each is the centre of an isotropic Gaussian kernel. The resulting regression function, given by the conditional mean, is shown by the red line, along with the two-standard-deviation region for the conditional distribution $p(t|x)$ shown by the red shading. The blue ellipse around each data point shows one standard deviation contour for the corresponding kernel. These appear noncircular due to the different scales on the horizontal and vertical axes.

Figure 0

## 4.2 Polynomial Kernels in Classification

One very common use for polynomial kernels is in classification. Let us look at the following example:
Assume we have a set of data points, labeled red or blue and want to separate them as we can see in the figure 1.1 below.

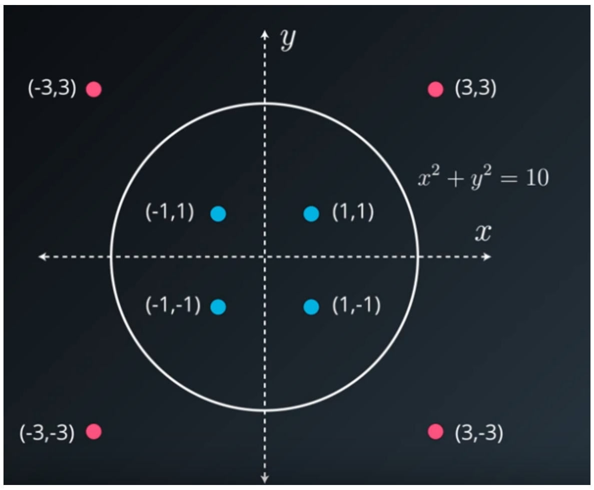

Figure 1.1

It is easy to see that we can not separate them with a linear function, but for instance with a circle.
To get a linear separator we need to go to a higher dimension, in this case dimension 3, where we can choose $z = x^2 + y^2$. The blue cone we see in figure 1.2 consists of all points that satisfy the equation $z = x^2 + y^2$, in short every data point from our original two dimensional space will be projected on this cone.
We can see in figure 1.3 that we can separate the two sets of data points by the linear separator $z = 10$ in this dimension. When we look a the intersection of this separator and the cone we get the same circle we drew in our original data space.
This polynomial transformation into a higher dimension and back always corresponds to a polynomial kernel and vice versa. In this instance we have the feature mapping $\phi((x, y)) = (x, y, x^2 + y^2)$ and the corresponding polynomial kernel $k((x, y), (x', y')) = \Phi((x, y))^T \Phi((x, y)) = xx' + yy' + x^2x'^2 + y^2y'^2 + x^2y'^2 + y^2x'^2$.

Obviously this is quite an easy example, where it is easy to compute the feature map. This is not always the case and can be computationally costly. But when we use kernels we do not have to compute this feature map explicitly, we just have to use the corresponding kernel in the Gram-Matrix, as we have seen in chapter 2 about dual representation. So if we have data points that are not linearly separable in the original data set, but we find a polynomial function like the circle to separate them, we know now it corresponds to a linear separator in a higher dimension and this is enough to compute the kernel, which is all that is needed.
In short the main advantage here is when we use algorithms that only depend on the Gram-matrix (denoted by G, which is defined by Kernel function), then we never have to know or even compute the actual feature map. For most of the cases, efficient algorithms exist since we know the kernel function.

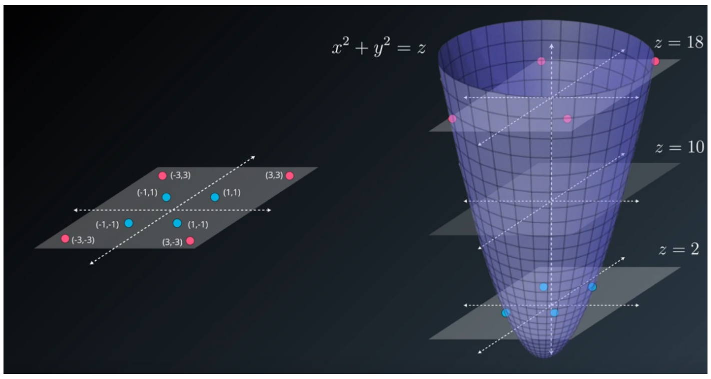

Figure 1.2

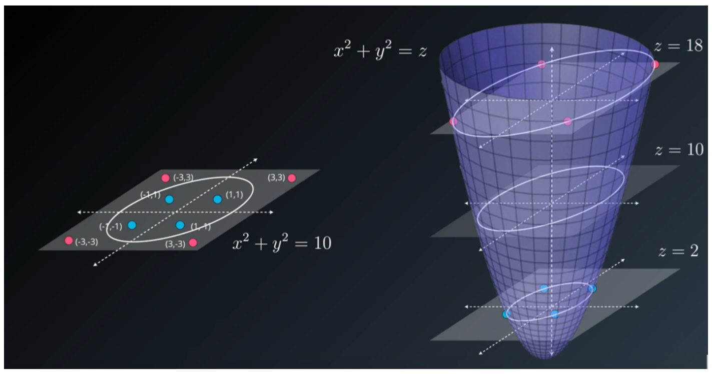

Figure 1.3

## 4.3 Graph kernels

As we stated in chapter 3, with kernel functions it is possible not just to have numerical inputs. Here we now have an example for such an event. For further detail on this example see [SNVB10].
Assume one wants to evaluate a protein. Proteins can be represented by certain forms of different graphs as seen in Figure 1.

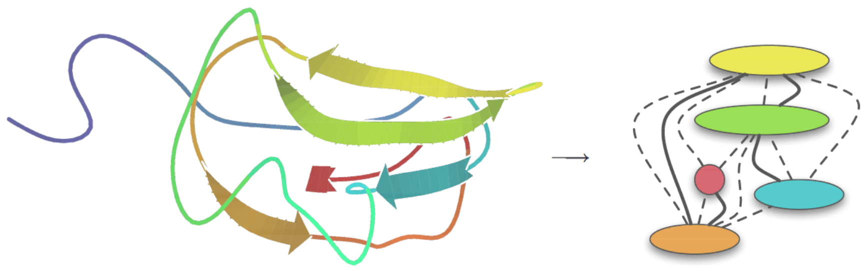

Figure 1: Left: Structure of *E. coli* protein fragment APO-BCCP87 (Yao et al., 1997), ID 1a6x in the Protein Data Bank (Berman et al., 2000). Right: Borgwardt et al.'s (2005) graph representation for this protein fragment. Nodes represent secondary structure elements, and edges encode neighborhood along the amino acid chain (solid) resp. in Euclidean 3D space (dashed).

We first have to define an inner product for graphs, and we can do this by using the Kronecker product. An example for this can be seen in Figure 2.

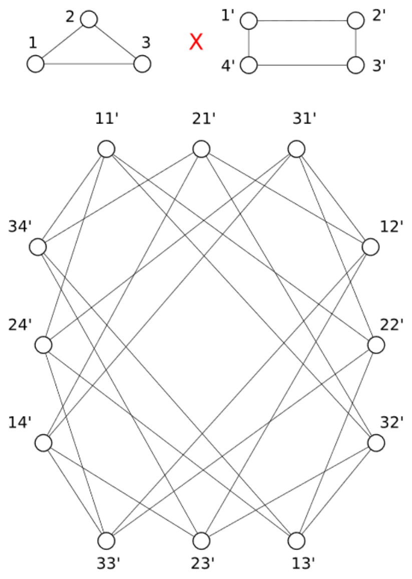

Figure 2: Two graphs (top left & right) and their direct product (bottom). Each node of the direct product graph is labeled with a pair of nodes (4); an edge exists in the direct product if and only if the corresponding nodes are adjacent in both original graphs (5). For instance, nodes $11'$ and $32'$ are adjacent because there is an edge between nodes 1 and 3 in the first, and $1'$ and $2'$ in the second graph.

Now We can define a kernel $k(G, G')$ for the graphs $G$ and $G'$ by using the corresponding weight function $W_x$ for the new graph $G_x = G \otimes G'$. We use the weight function to compute $q_x^T W_x^k p_x$, which is the expected similarity between random walks of same length k on $G$ and $G'$. The initial and stopping probability distributions are given by $p_x$ and $q_x$. We can now define the kernel by summing up the similarities of walks of all lengths. To make sure the sum converges we can add an appropriate non-negative coefficient $\mu(k)$ and therefore the kernel between $G$ and $G'$ can be defined as

$$k(G, G') := \sum_{k=0}^{\infty} \mu(k) q_x^T W_x^k p_x. \tag{31}$$

## References

[AIQ19] AIQCAR: *12 Support Vector Machine(SVM) Polynomial Kernel Detail Explanation.* 2019 `https://www.youtube.com/watch?v=Xoz3LeOWOGU`

[Bis06] Bishop, Christopher M.: *Pattern recognition and machine learning.* New York : Springer Science and Business Media,LLC, 2006. – 291–303 S.

[PV] Philippe Vert, Bernhard S. Koji Tsuda T. Koji Tsuda: *A primer on kernel methods*

[SNVB10] S.V. N. Vishwanathan, Risi K. Nicol N. Schraudolph S. Nicol N. Schraudolph ; Borgwardt, Karsten M.: *Graph Kernels.* Journal of Machine Learning Research 11 (2010), 2010 `https://jmlr.csail.mit.edu/papers/volume11/vishwanathan10a/vishwanathan10a.pdf`