Machine Learning

SVMs

Manfred Jaeger

Aalborg University

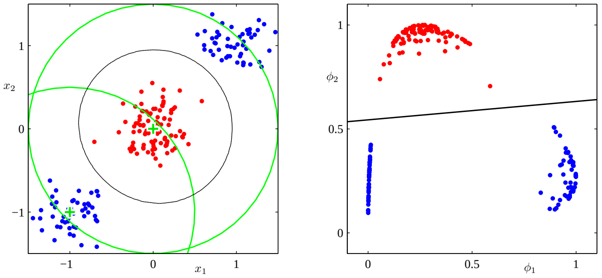
Data Transformations

Any mapping

$$\phi : \mathbb{R}^D \rightarrow \mathbb{R}^{D'}$$

defines a data transformation that transforms the original data instance $\boldsymbol{x}$ into a transformed data instance $\phi(\boldsymbol{x}) = (\phi_1(\boldsymbol{x}), \ldots, \phi_{D'}(\boldsymbol{x}))$.

- The components $\phi_i(\boldsymbol{x})$ are also called **features** or **basis functions**, and $\mathbb{R}^{D'}$ the **feature space** of ϕ.
- Often: $D' > D$, and ϕ_i typically non-linear.



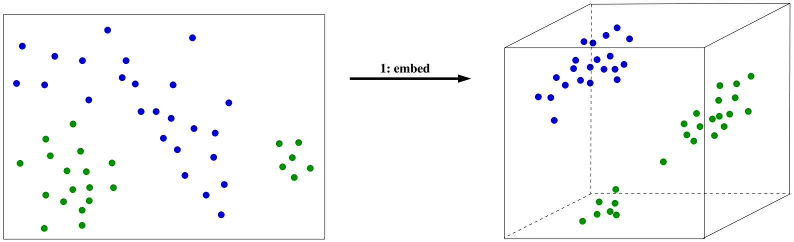
[Bishop Fig. 4.12]

Example: Original data (left), transformed data in feature space (right), decision boundaries (black lines). Here: $\phi_1(x_1, x_2) = \exp(-((x_1 + 1)^2 + (x_2 + 1)^2))$ and $\phi_2(x_1, x_2) = \exp(-(x_1^2 + x_2^2))$.

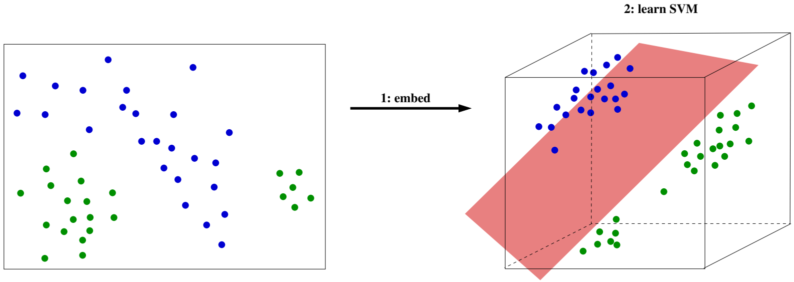
- All our linear models can be applied to the transformed data
- The linear decision boundary in feature space then corresponds to a non-linear decision boundary in the original space

Nonlinear SVM

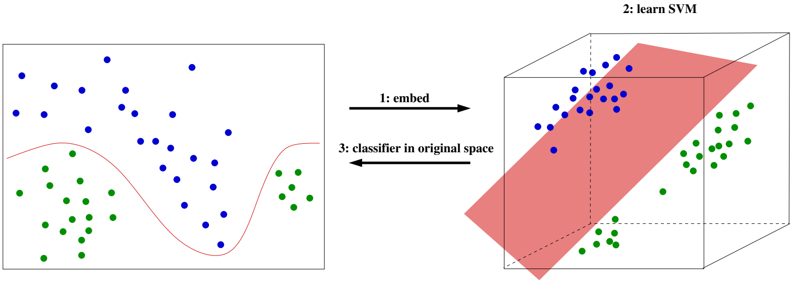
SVM learning after a non-linear transformation:



SVM learning after a non-linear transformation:



SVM learning after a non-linear transformation:



Problem: suitable feature spaces may need to be very high-dimensional. That makes computations with transformed vectors $\phi(\boldsymbol{x})$ very expensive.

The SVM learning problem is solved using the method of Lagrange multipliers:

- In the Lagrange optimization process: to determine the support vectors $\boldsymbol{x}_i$, the λ_i and b: the only operations required on data items is to compute dot products $\boldsymbol{x}_i \cdot \boldsymbol{x}_j$.
- for classification: only need to compute dot products $\boldsymbol{x}_i \cdot \boldsymbol{z}$

A dot product $\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{x}')$ may be computed without actually constructing the (high-dimensional) vectors $\phi(\boldsymbol{x}), \phi(\boldsymbol{x}')$.

Example

$$\phi : \ (x_1, x_2) \to (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2, 1)$$

Then

$$\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{x}') = x_1^2 {x_1'}^2 + x_2^2 {x_2'}^2 + 2x_1 x_1' + 2x_2 x_2' + 2x_1 x_1' x_2 x_2' + 1 = (\boldsymbol{x} \cdot \boldsymbol{x}' + 1)^2$$

- $K(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x} \cdot \boldsymbol{x}' + 1)^2$ is an example of a **kernel function**
- Kernel functions (short: kernels) can be interpreted as measures of similarity

Conversely, let

$$K(\boldsymbol{x}, \boldsymbol{z})$$

be a a symmetric (Kernel) function ($K(\boldsymbol{x}, \boldsymbol{z}) = K(\boldsymbol{z}, \boldsymbol{x})$) function that measures similarity between $\boldsymbol{x}$ and $\boldsymbol{z}$.

Can we interpret $K(\boldsymbol{x}, \boldsymbol{z})$ as a dot product $\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{z})$ in some feature space?

Mercer's Theorem

A kernel $K(\boldsymbol{x}, \boldsymbol{x}')$ is of the form $\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{x}')$ if and only if for all functions $g(\boldsymbol{x})$ with $\int g(\boldsymbol{x})^2 d\boldsymbol{x} < \infty$:

$$\int K(\boldsymbol{x}, \boldsymbol{x}') g(\boldsymbol{x}) g(\boldsymbol{x}') d\boldsymbol{x} d\boldsymbol{x}' \geq 0$$

(K then is called **positive semi-definite**).

From Kernels to Dot Products II

Let $K(\boldsymbol{x},\boldsymbol{z})$ be a symmetric (Kernel) function and $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n$ a set of datapoints.

Then

$$\begin{pmatrix} K(\boldsymbol{x}_1,\boldsymbol{x}_1) & \ldots & K(\boldsymbol{x}_1,\boldsymbol{x}_j) & \ldots & K(\boldsymbol{x}_1,\boldsymbol{x}_n) \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ K(\boldsymbol{x}_i,\boldsymbol{x}_1) & \ldots & K(\boldsymbol{x}_i,\boldsymbol{x}_j) & \ldots & K(\boldsymbol{x}_i,\boldsymbol{x}_n) \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ K(\boldsymbol{x}_n,\boldsymbol{x}_1) & \ldots & K(\boldsymbol{x}_n,\boldsymbol{x}_j) & \ldots & K(\boldsymbol{x}_n,\boldsymbol{x}_n) \end{pmatrix}$$

is the (symmetric) *Kernel matrix* (also called *Gram matrix*) for $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n$

Matrix Version of Mercer's Theorem

A symmetric function $K(\boldsymbol{x},\boldsymbol{z})$ is of the form

$$K(\boldsymbol{x},\boldsymbol{z}) = \phi(\boldsymbol{x})\cdot\phi(\boldsymbol{z})$$

for some feature mapping ϕ, if and only if for all finite sets of points $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n$ the kernel matrix is *positive semi-definite*.

Eigenvalues

A $n \times n$ matrix M defines a linear transformation of $\mathbb{R}^n$:

$$\boldsymbol{x} \mapsto M\boldsymbol{x}$$

A vector $\boldsymbol{x}$ is an **eigenvector** of M if

$$M\boldsymbol{x} = \lambda\boldsymbol{x}$$

for some constant (=**eigenvalue**) $\lambda \in \mathbb{R}$. A matrix is **positive-semidefinite**, if all of its eigenvalues are non-negative.

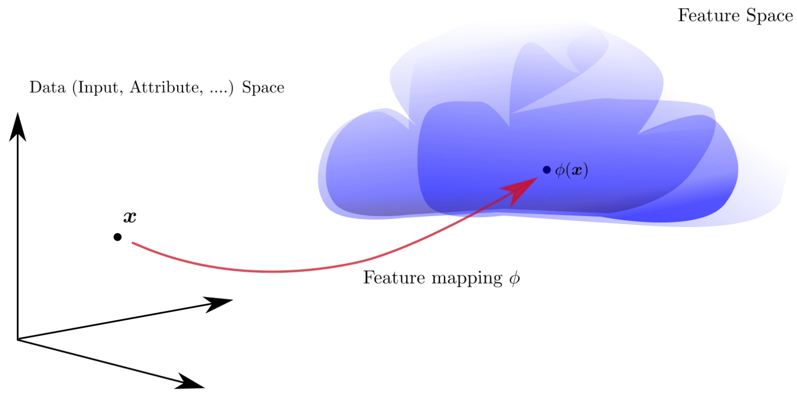
Quadratic Forms

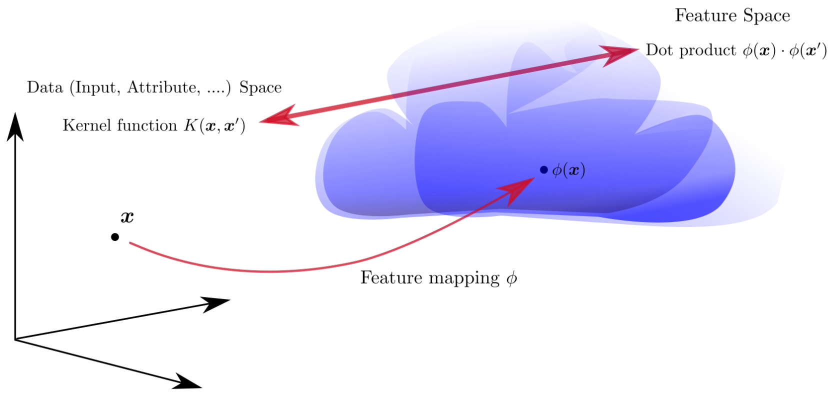
A matrix also defines the function

$$\boldsymbol{x} \mapsto \boldsymbol{x}^T M\boldsymbol{x} = \sum_{i,j} x_i x_j M_{i,j} \in \mathbb{R}.$$

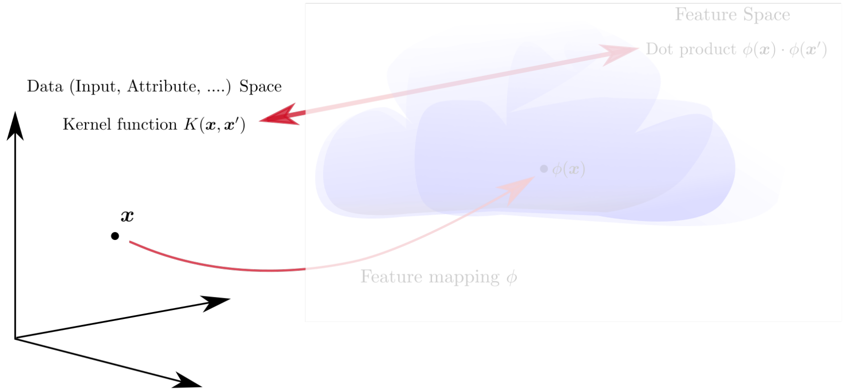
Positive semi-definiteness is also characterized by

$$\boldsymbol{x}^T M\boldsymbol{x} \geq 0 \quad \text{for all} \quad \boldsymbol{x} \in \mathbb{R}^n$$


Feature Space
Data (Input, Attribute, ...) Space
$\boldsymbol{x}$
$\phi(\boldsymbol{x})$
Feature mapping ϕ


Feature Space
Dot product $\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{x}')$
Data (Input, Attribute, ...) Space
Kernel function $K(\boldsymbol{x}, \boldsymbol{x}')$
$\phi(\boldsymbol{x})$
$\boldsymbol{x}$
Feature mapping ϕ

Feature Space and Kernels


Feature Space
Dot product $\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{x}')$
Data (Input, Attribute, ...) Space
Kernel function $K(\boldsymbol{x}, \boldsymbol{x}')$
$\phi(\boldsymbol{x})$
$\boldsymbol{x}$
Feature mapping ϕ

➥When we have a posititive semi-definite kernel $K(\boldsymbol{x}, \boldsymbol{x}')$ we need not know the actual feature mapping ϕ in order to perform computations involving dot products $\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{x}')$.

We obtain the following strategy to construct *Kernelized* SVM classifiers:

- **Given:** a classification problem
- **Define** or **Select** a similarity function
$$K(\boldsymbol{x}, \boldsymbol{z})$$
- **Verify** that $K(\boldsymbol{x}, \boldsymbol{z})$ is positive semi-definite
- **Learn** an SVM using the values $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ in place of dot products $\boldsymbol{x}_i \cdot \boldsymbol{x}_j$.

➥The learning algorithm requires as input "only" the kernel matrix for the training datapoints $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$.

Basic Kernels

$(\boldsymbol{x} \cdot \boldsymbol{z} + 1)^p$	Polynomial Kernel (sklearn: `poly`)
$e^{-\|\boldsymbol{x}-\boldsymbol{z}\|/2\sigma^2}$	Gaussian kernel (sklearn: `rbf`)
$\tanh(\kappa \cdot \boldsymbol{x} \cdot \boldsymbol{z} - \delta))$	Hyperbolic tangent (sklearn: `sigmoid`)

Kernel Building Rules

If $K(\boldsymbol{x}, \boldsymbol{x}')$ is a positive semi-definite kernel, then so is

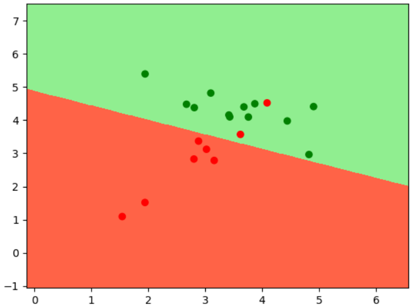
- $q(K(\boldsymbol{x}, \boldsymbol{x}'))$, where $q()$ is a polynomial with nonnegative coefficients
- $e^{K(\boldsymbol{x}, \boldsymbol{x}')}$
- $K(\boldsymbol{x}, \boldsymbol{x}')K'(\boldsymbol{x}, \boldsymbol{x}')$, where $K'()$ is another positive definite kernel
- ...
- $\frac{K(\boldsymbol{x}, \boldsymbol{x}')}{\sqrt{K(\boldsymbol{x}, \boldsymbol{x})K(\boldsymbol{x}', \boldsymbol{x}')}}$: the **Normalization** of $K()$.

Implicit assumption so far: data linearly separable, maybe after kernel transformation.

In reality: unlikely to be true

Example

The scikit-learn model `SVC(kernel='linear')` somehow also handles data that is not linearly separable:



How does it do that?

Slack variables

We relax the obtimization objective by introducing *slack variables* ζ_n $(n \in \{1, \ldots, N\})$:

Original: minimize

$$\frac{1}{2} \parallel \boldsymbol{w} \parallel^2$$

subject to

$$y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) \geq 1$$

Relaxed: minimize

$$\frac{1}{2} \parallel \boldsymbol{w} \parallel^2 + C \sum_{n=1}^{N} \zeta_n$$

subject to

$$y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) \geq 1 - \zeta_n \quad (n = 1, \ldots, N).$$

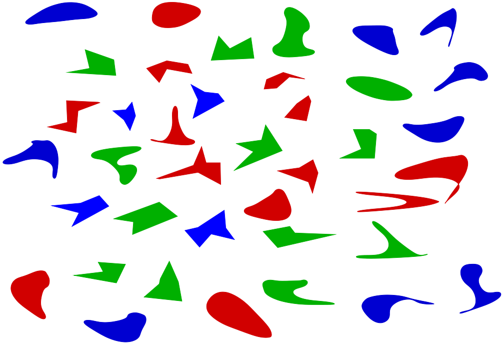
- the ζ_n are constrained to be non-negative
- the hyper-parameter C controls how much the loss increases when we relax the constraints with $\zeta_n > 0$.
- for linearly separable data: small values of C can lead to solutions that do not separate the data

Kernels for Non Standard Data

So far: have assumed that datapoints $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ lie in $\mathbb{R}^D$.

Kernels can be defined, and SVM classifiers learned, for data from very different spaces.

Datapoints consisting of colored geometric shapes:



Define a kernel function $K(shape_i, shape_j)$, and verify that the kernel matrices will be positive semi-definite:

			Train			*Test*
	1	-1	-1	-1	1	
	1	0.4	0.1	0.6	0.5	0.5
	0.4	1	0.6	0.4	0.4	0.5
	0.1	0.6	1	0.3	0.1	0.1
	0.6	0.4	0.3	1	0.2	0.4
	0.5	0.4	0.1	0.2	1	0.5

(values here are hypothetical values computed by some imaginary kernel).

➥Can now learn SVM from *Train* kernel matrix, and classify test case(s) using the *Train*$\times$ *Test* kernel matrix.

Movie Reviews:

> *There is a level of high expectation when you sit down to watch a comedy with a cast headed by Cary Grant, Jayne Mansfield, Ray Walston and Werner Klemperer. Those expectations are buoyed further when the film is directed by Stanley Donen, whose comic touch was so evident in, among others, DAMN YANKEES!, BEDAZZLED and CHARADE. For the first five minutes, or so, it seems that those expectations might be met and then. Nothing. What is supposed to be a light comedy, plunges into leaden, heavy handed melodrama,. . .*
>
> *Everyone plays their part pretty well in this "little nice movie". Belushi gets the chance to live part of his life differently, but ends up realizing that what he had was going to be just as good or maybe even better. The movie shows us that we ought to take advantage of the opportunities we have, not the ones we do not or cannot have.. . .*

Classification task: does a review express a *positive* or *negative* sentiment?

Term frequency vector

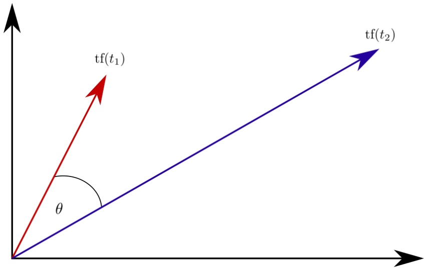
After some preprocessing (putting everything in lower case, omitting punctuation, stemming) the text becomes a sequence of **terms** from a fixed **vocabulary**.

For a vocabulary of size n (e.g. $n = 100.000$) represent a text t by the n-dimensional vector $\mathrm{tf}(t)$:

$$\mathrm{tf}(t)[i] = \text{number of times the } i\text{th term of the dictionary appears in } t$$

➡The order of words is not encoded in $\mathrm{tf}(t)$: **bag of words** model of text.

➡Since most components of $\mathrm{tf}(t)$ are zero, should use a sparse representation.



- If t_1 and t_2 do not have any terms in common, then their tf vectors are orthogonal ($cosine(\theta)$=0).
- If t_1 and t_2 contain exactly the same terms with the same relative frequencies, then $\mathrm{tf}(t_1) = r \cdot \mathrm{tf}(t_2)$ for some constant r, and $cosine(\theta)$=1. (E.g.: t_1 is the concatenation of two copies of t_2, and $r = 2$).

Cosine similarity

$$cos\text{-}sim(t_1, t_2) = cosine(\theta) = \frac{\mathrm{tf}(t_1) \cdot \mathrm{tf}(t_2)}{\| \mathrm{tf}(t_1) \| \cdot \| \mathrm{tf}(t_2) \|}$$

Kernel

cos-sim is a positive semi-definite kernel: normalization of plain dot product (because $\| \boldsymbol{x} \| = \sqrt{\boldsymbol{x} \cdot \boldsymbol{x}}$).

No kernel trick here: we work with explicitly constructed feature vectors $\mathrm{tf}(t)$.

Information Retrieval

cos-sim is extensively used in information retrieval as a measure of similarity between documents:

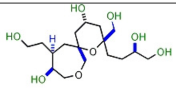
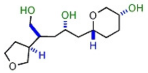
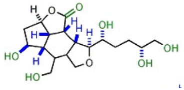
- t_1: a short query text
- t_2: a candidate document that may be returned for query t_1.

Strings

Biological sequences:

$$
\begin{array}{ccccccccccccccc}
\ldots & A & A & C & G & A & A & T & C & A & A & C & C & A & \ldots \\
 & | & | & | & | & | & | & | & | & | & | & | & | & | & \\
\ldots & T & T & G & C & T & T & A & G & T & T & G & G & T & \ldots
\end{array}
$$

Graphs

Instance	Data	*Class*
1		0
2		1
3		0

A data instance has the form

$$(\boldsymbol{s}_n, c_n)$$

with

- $\boldsymbol{s}_n \in \Sigma^*$ for some alphabet Σ
- $c_n \in \{0, 1\}$ (class label)

Basic String Features

For any $\boldsymbol{u} \in \Sigma^*$:

$$\begin{aligned} \phi_{\boldsymbol{u}}(\boldsymbol{s}) &:= \quad \#\text{occurrences of } \boldsymbol{u} \text{ as a } \textbf{substring} \text{ of } \boldsymbol{s} \\ \phi^+_{\boldsymbol{u}}(\boldsymbol{s}) &:= \quad \#\text{occurrences of } \boldsymbol{u} \text{ as a } \textbf{subsequence} \text{ of } \boldsymbol{s} \end{aligned}$$

Example

$$\boldsymbol{s} = \textit{statistics}$$

	$\boldsymbol{u}$		
	ti	*tis*	*atics*
$\phi_{\boldsymbol{u}}(\boldsymbol{s})$	2	1	0
$\phi^+_{\boldsymbol{u}}(\boldsymbol{s})$	5	7	3

Defined by the feature vector

$$\{\phi_{\boldsymbol{u}} \mid \boldsymbol{u} \in \Sigma^p\}$$

Example [Shawe-Taylor, Cristianini, p. 348]

2-spectrum feature vectors for $\boldsymbol{s}, \boldsymbol{t} \in \{bar, bat, car, cat\}$ (with $\Sigma = \{a, b, \ldots, z\}$):

	aa	ab	...	ar	...	at	...	ba	...	ca	...	zz
ϕ(bar)	0	0	...	1	...	0	...	1	...	0	...	0
ϕ(bat)	0	0	...	0	...	1	...	1	...	0	...	0
ϕ(car)	0	0	...	1	...	0	...	0	...	1	...	0
ϕ(cat)	0	0	...	0	...	1	...	0	...	1	...	0

Gives the kernel matrix:

	bar	bat	car	cat
bar	2	1	1	0
bat	1	2	0	1
car	1	0	2	1
cat	0	1	1	2

Defined by the feature vector

$$\{\phi^+_{\boldsymbol{u}} \mid \boldsymbol{u} \in \Sigma^*\}$$

For the empty string ϵ define $\phi^+_\epsilon(\boldsymbol{s}) = 1$ for all $\boldsymbol{s}$.

Example [Shawe-Taylor, Cristianini, p. 352]

Values of all-subsequence kernel for $\boldsymbol{s}, \boldsymbol{t} \in \{bar, baa, car, cat\}$.

	bar	baa	car	cat
bar	8	6	4	2
baa	6	12	3	3
car	4	3	8	4
cat	2	3	4	8

	ϵ	a	b	...	r	...	aa	ab	...	ba	...	br	...	baa	...	bar	...
ϕ^+(bar)	1	1	1		1		0	0		1		1		0		1	
ϕ^+(baa)	1	2	1		0		1	0		2		0		1		0	

- Cannot compute whole feature vector (infinite!)
- Explicit computation of all *nonzero* components of feature vector $\phi(\boldsymbol{s})$ also infeasible (exponential in the length of $\boldsymbol{s}$)
- Using a dynamic programming approach, one can compute $K(\boldsymbol{s}, \boldsymbol{t})$ in time $|\boldsymbol{s}| \cdot |\boldsymbol{t}|$.

Dynamic Programming Approach

- $\boldsymbol{s} = s_1 \dots s_n$, $\boldsymbol{t} = t_1 \dots t_m$
- For $\boldsymbol{i} \subseteq \{1, \dots, n\}$: $\boldsymbol{s}[\boldsymbol{i}]$: subsequence defined by $\boldsymbol{i}$
- $$K(\boldsymbol{s}, \boldsymbol{t}) = \#(\boldsymbol{i}, \boldsymbol{j}) : \boldsymbol{s}[\boldsymbol{i}] = \boldsymbol{t}[\boldsymbol{j}]$$

Recursion:

$$\begin{aligned} K(\epsilon, \boldsymbol{t}) &= 1 \\ K(\boldsymbol{s}[1:i], \boldsymbol{t}[1:j]) &= K(\boldsymbol{s}[1:i-1], \boldsymbol{t}[1:j]) + \sum_{k \le j : t_k = s_i} K(\boldsymbol{s}[1:i-1], \boldsymbol{t}[1:k-1]) \end{aligned}$$

Count of common sub-sequences not involving s_i
Count of common sub-sequences involving s_i

- Straightforward implementation: $O(nm^2)$
- With added tricks and optimizations can compute $K(\boldsymbol{s}, \boldsymbol{t})$ in $O(nm)$.

SVM + Kernel Functions:

- + Powerful method for classification
- + Successful applications, e.g. in bio-informatics
- + Wide variety of data types and classification problems reduced to a single type of optimization problem
- – Binary classifier only: generalization to multiclass only via multipe binary classifiers
- – Complexity quadratic in number of instances
- – To find the “right” kernel function may require a lot of engineering