Kernels and the Kernel Trick
Support vector machines

- Training by maximizing margin
- The SVM objective
- Solving the SVM optimization problem
- Support vectors, duals and kernels
Support vector machines

• Training by maximizing margin

• The SVM objective

• Solving the SVM optimization problem

• Support vectors, duals and kernels
This lecture

1. Support vectors

2. Kernels

3. The kernel trick

4. Properties of kernels

5. Another example of the kernel trick
This lecture

1. Support vectors

2. Kernels

3. The kernel trick

4. Properties of kernels

5. Another example of the kernel trick
So far we have seen

- Support vector machines
- Hinge loss and optimizing the regularized loss

More broadly, different algorithms for learning linear classifiers
So far we have seen

- Support vector machines
- Hinge loss and optimizing the regularized loss

More broadly, different algorithms for learning linear classifiers

What about non-linear models?
One way to learn non-linear models

Explicitly introduce non-linearity into the feature space

If the true separator is quadratic
One way to learn non-linear models

Explicitly introduce non-linearity into the feature space

If the true separator is quadratic

Transform all input points as

\[
\phi(x_1, x_2) = \begin{bmatrix}
x_1 \\
x_2 \\
x_1^2 \\
x_2^2
\end{bmatrix}
\]
One way to learn non-linear models

Explicitly introduce non-linearity into the feature space

If the true separator is quadratic

Transform all input points as

\[
\phi(x_1, x_2) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \end{bmatrix}
\]

Now, we can try to find a weight vector in this higher dimensional space

That is, predict using \( w^T \phi(x_1, x_2) \geq b \)
SVM: Primals and duals

The SVM objective

\[
\min_{w, \xi} \quad \frac{1}{2} w^T w + C \sum_i \xi_i \\
\text{s.t.} \quad \forall i, \quad y_i w^T x_i \geq 1 - \xi_i \\
\forall i, \quad \xi_i \geq 0.
\]

This is called the *primal form* of the objective.

This can be converted to its *dual form*, which will let us prove a very useful property.
SVM: Primals and duals

The SVM objective

\[
\begin{align*}
\min_{\mathbf{w}, \xi} & \quad \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_i \xi_i \\
\text{s.t.} & \quad \forall i, \, y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i \\
& \quad \forall i, \, \xi_i \geq 0.
\end{align*}
\]

Another optimization problem

Has the property that \( \max \text{Dual} = \min \text{Primal} \)

This is called the \textit{primal form} of the objective

This can be converted to its \textit{dual form}, which will let us prove a very useful property.
Support vector machines

Let $\mathbf{w}$ be the minimizer of the SVM problem for some dataset with $m$ examples: $\{(\mathbf{x}_i, y_i)\}$

Then, for $i = 1 \ldots m$, there exist $\alpha_i \geq 0$ such that the optimum $\mathbf{w}$ can be written as

$$
\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i
$$
Support vector machines

Let \( \mathbf{w} \) be the minimizer of the SVM problem for some dataset with m examples: \( \{(\mathbf{x}_i, y_i)\} \)
Then, for \( i = 1 \ldots m \), there exist \( \alpha_i \geq 0 \) such that the optimum \( \mathbf{w} \) can be written as

\[
\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i
\]

Furthermore, \( \alpha_i = 0 \)

\[
\Rightarrow y_i \mathbf{w}^T \mathbf{x}_i \geq 1
\]

All points outside the margin
Support vector machines

Let $\mathbf{w}$ be the minimizer of the SVM problem for some dataset with $m$ examples: \{$(\mathbf{x}_i, y_i)$\}

Then, for $i = 1 \ldots m$, there exist $\alpha_i \geq 0$ such that the optimum $\mathbf{w}$ can be written as

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

Furthermore,

$\alpha_i = 0 \quad \Rightarrow \quad y_i \mathbf{w}^T \mathbf{x}_i \geq 1$

$\alpha_i = C \quad \Rightarrow \quad y_i \mathbf{w}^T \mathbf{x}_i \leq 1$

All points on the wrong side of the margin
Support vector machines

Let $\mathbf{w}$ be the minimizer of the SVM problem for some dataset with $m$ examples: $\{(\mathbf{x}_i, y_i)\}$

Then, for $i = 1 \ldots m$, there exist $\alpha_i \geq 0$ such that the optimum $\mathbf{w}$ can be written as

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

Furthermore,

$$\alpha_i = 0$$  $\Rightarrow$  $$y_i \mathbf{w}^T \mathbf{x}_i \geq 1$$

$$\alpha_i = C$$  $\Rightarrow$  $$y_i \mathbf{w}^T \mathbf{x}_i \leq 1$$

$$0 \leq \alpha_i \leq C$$  $\Rightarrow$  $$y_i \mathbf{w}^T \mathbf{x}_i = 1$$

All points on the margin
Support vectors

The weight vector is completely defined by training examples whose $\alpha_i$s are not zero

\[
\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i
\]

These examples are called the support vectors
This lecture

✓ Support vectors

2. Kernels

3. The kernel trick

4. Properties of kernels

5. Another example of the kernel trick
Predicting with linear classifiers

• Prediction $= \text{sgn}(\mathbf{w}^T \mathbf{x})$ and $\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$
Predicting with linear classifiers

- Prediction: $\text{sgn}(\mathbf{w}^T \mathbf{x})$ and $\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$

- That is, we just showed that $\mathbf{w}^T \mathbf{x} = \sum_{i} \alpha_i y_i \mathbf{x}_i^T \mathbf{x}$

  - We only need to compute dot products between training examples and the new example $\mathbf{x}$
Predicting with linear classifiers

• Prediction = $\text{sgn}(w^T x)$ and $w = \sum_{i=1}^{m} \alpha_i y_i x_i$

• That is, we just showed that $w^T x = \sum_{i} \alpha_i y_i x_i^T x$

– We only need to compute dot products between training examples and the new example $x$
Predicting with linear classifiers

• Prediction = \( sgn(\mathbf{w}^T \mathbf{x}) \) and \( \mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \)

• That is, we just showed that \( \mathbf{w}^T \mathbf{x} = \sum_{i} \alpha_i y_i \mathbf{x}_i^T \mathbf{x} \)

  – We only need to compute dot products between training examples and the new example \( \mathbf{x} \)

• This is true even if we map examples to a high dimensional space

  \[
  \mathbf{w}^T \phi(\mathbf{x}) = \sum_{i} \alpha_i y_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x})
  \]
Predicting with linear classifiers

- Prediction = ... and

- That is, we just showed that
  - That is we only need to compute dot products between training examples and the new example
  - This is true even if we map examples to a high dimensional space

One way to learn non-linear models

Explicitly introduce non-linearity into the feature space

If the true separator is quadratic

Transform all input points as

\[ \phi(x_1, x_2) = \begin{bmatrix} x_1 \\ x_2 \\ x^2_1 \\ x^2_2 \end{bmatrix} \]

Now, we can try to find a weight vector in this higher dimensional space

That is, predict using \( \mathbf{w}^T \phi(x_1, x_2) \geq b \)
Dot products in high dimensional spaces

Let us define a dot product in the high dimensional space

\[ K(x, z) = \phi(x)^T \phi(z) \]
Dot products in high dimensional spaces

Let us define a dot product in the high dimensional space

$$K(x, z) = \phi(x)^T \phi(z)$$

So prediction with this high dimensional lifting map is

$$\text{sgn}(w^T \phi(x)) = \text{sgn} \left( \sum_i \alpha_i y_i K(x_i, x) \right)$$
Dot products in high dimensional spaces

Let us define a dot product in the high dimensional space

\[ K(x, z) = \phi(x)^T \phi(z) \]

So prediction with this *high dimensional lifting map* is

\[ \text{sgn}(w^T \phi(x)) = \text{sgn} \left( \sum_i \alpha_i y_i K(x_i, x) \right) \]

because \( w^T \phi(x) = \sum_i \alpha_i y_i \phi(x_i)^T \phi(x) \)
Kernel based methods

\[ K(x, z) = \phi(x)^T \phi(z) \]

Predict using \[ \text{sgn}(w^T \phi(x)) = \text{sgn} \left( \sum_i \alpha_i y_i K(x_i, x) \right) \]

What does this new formulation give us?

If we have to compute \( \phi \) every time anyway, we gain nothing
Kernel based methods

$$K(x, z) = \phi(x)^T \phi(z)$$

Predict using

$$\text{sgn}(w^T \phi(x)) = \text{sgn} \left( \sum_i \alpha_i y_i K(x_i, x) \right)$$

What does this new formulation give us?

If we have to compute $\phi$ every time anyway, we gain nothing.

If we can compute the value of $K$ without explicitly writing the blown up representation, then we will have a computational advantage.
This lecture

- Support vectors
- Kernels

3. The kernel trick

4. Properties of kernels

5. Another example of the kernel trick
Example: Polynomial Kernel

- Given two examples $x$ and $z$ we want to map them to a high dimensional space [for example, quadratic]

$$
\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1x_2, \cdots, x_{n-1}x_n]^T
$$
Example: Polynomial Kernel

- Given two examples $x$ and $z$ we want to map them to a high dimensional space [for example, quadratic]

\[
\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1x_2, \cdots, x_{n-1}x_n]^T
\]

All degree zero terms
Example: Polynomial Kernel

- Given two examples $x$ and $z$ we want to map them to a high dimensional space [for example, quadratic]

\[
\phi(x_1, x_2, \ldots, x_n) = [1, x_1, x_2, \ldots, x_n, x_1^2, x_2^2, \ldots, x_n^2, x_1x_2, \ldots, x_{n-1}x_n]^T
\]

All degree zero terms  All degree one terms
Example: Polynomial Kernel

- Given two examples \( x \) and \( z \) we want to map them to a high dimensional space [for example, quadratic]

\[
\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1x_2, \cdots, x_{n-1}x_n]^T
\]
Example: Polynomial Kernel

- Given two examples \( x \) and \( z \) we want to map them to a high dimensional space [for example, quadratic]

\[
\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1 x_2, \cdots, x_{n-1} x_n]^T
\]

All degree zero terms  All degree one terms  All degree two terms

and compute the dot product \( A = \phi(x)^T \phi(z) \)  [takes time ]
Example: Polynomial Kernel

• Given two examples $x$ and $z$ we want to map them to a **high dimensional space** [for example, quadratic]

$$
\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1x_2, \cdots, x_{n-1}x_n]^T
$$

and compute the dot product $A = \phi(x)^T \phi(z)$ [takes time]
Example: Polynomial Kernel

• Given two examples \( x \) and \( z \) we want to map them to a high dimensional space [for example, quadratic]

\[
\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1x_2, \cdots, x_{n-1}x_n]^T
\]

and compute the dot product \( A = \phi(x)^T \phi(z) \) [takes time]

• Instead, in the original space, compute
Example: Polynomial Kernel

- Given two examples $x$ and $z$ we want to map them to a high dimensional space [for example, quadratic]

$$
\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1x_2, \cdots, x_{n-1}x_n]^T
$$

and compute the dot product $A = \phi(x)^T \phi(z)$ [takes time]

- Instead, in the original space, compute

$$
B = K(x, z) = (1 + x^T z)^2
$$
Example: Polynomial Kernel

• Given two examples $x$ and $z$ we want to map them to a high dimensional space [for example, quadratic]

$$
\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1 x_2, \cdots, x_{n-1} x_n]^T
$$

and compute the dot product $A = \phi(x)^T \phi(z)$ [takes time ]

• Instead, in the original space, compute

$$
B = K(x, z) = (1 + x^T z)^2
$$

Claim: $A = B$ (Coefficients do not really matter)
Example: Two dimensions, quadratic kernel

\[ A = \phi(x)^T \phi(z) \quad \quad B = K(x, z) = (1 + x^T z)^2 \]

\[ \phi(x_1, x_2) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \\ x_1 x_2 \end{bmatrix} \]
The Kernel Trick

Suppose we wish to compute $K(x, z) = \phi(x)^T\phi(z)$

Here $\phi$ maps $x$ and $z$ to a high dimensional space.

*The Kernel Trick:* Save time/space by computing the value of $K(x, z)$ by performing operations in the original space (without a feature transformation!)
Computing dot products efficiently

**Kernel Trick:** You want to work with degree 2 polynomial features, $\phi(x)$. Then, your dot product will be operate using vectors in a space of dimensionality $n(n+1)/2$.

The kernel trick allows you to save time/space and compute dot products in an $n$ dimensional space.

*(Not just for degree 2 polynomials)*
This lecture

✓ Support vectors

✓ Kernels

✓ The kernel trick

4. Properties of kernels

5. Another example of the kernel trick
Which functions are kernels?

**Kernel Trick:** You want to work with degree 2 polynomial features, $\phi(x)$. Then, your dot product will be operate using vectors in a space of dimensionality $n(n+1)/2$.

The kernel trick allows you to save time/space and compute dot products in an $n$ dimensional space. *(Not just for degree 2 polynomials)*

- Can we use any function $K(.,.)$?
Which functions are kernels?

**Kernel Trick:** You want to work with degree 2 polynomial features, \( \phi(x) \). Then, your dot product will be operate using vectors in a space of dimensionality \( n(n+1)/2 \).

The kernel trick allows you to save time/space and compute dot products in an \( n \) dimensional space. *(Not just for degree 2 polynomials)*

- **Can we use any function \( K(.,..) \)?**
  - No! A function \( K(x,z) \) is a valid kernel if it corresponds to an inner product in some (perhaps infinite dimensional) feature space.
Which functions are kernels?

Kernel Trick: You want to work with degree 2 polynomial features, $\phi(x)$. Then, your dot product will be operate using vectors in a space of dimensionality $n(n+1)/2$.

The kernel trick allows you to save time/space and compute dot products in an $n$ dimensional space.

(Not just for degree 2 polynomials)

- Can we use any function $K(.,.)$?
  - No! A function $K(x,z)$ is a valid kernel if it corresponds to an inner product in some (perhaps infinite dimensional) feature space.

- General condition: construct the Gram matrix $\{K(x_i,z_j)\}$; check that it’s positive semi definite
Reminder: Positive semi-definite matrices

A symmetric matrix $M$ is positive semi-definite if it is

- For any vector non-zero $z$, we have $z^TMz \geq 0$

(A useful property characterizing many interesting mathematical objects)
The Kernel Matrix

• The **Gram matrix** of a set of $n$ vectors $S = \{x_1...x_n\}$ is the $n \times n$ matrix $G$ with $G_{ij} = x_i^T x_j$
  – The kernel matrix is the Gram matrix of $\{\phi(x_1), ..., \phi(x_n)\}$
  – (size depends on the # of examples, not dimensionality)
The Kernel Matrix

• The **Gram matrix** of a set of $n$ vectors $S = \{x_1...x_n\}$ is the $n \times n$ matrix $G$ with $G_{ij} = x_i^T x_j$
  
  – The kernel matrix is the Gram matrix of $\{\phi(x_1), ..., \phi(x_n)\}$
  
  – (size depends on the # of examples, not dimensionality)

• Showing that a function $K$ is a valid kernel
  
  – Direct approach: If you have the $\phi(x_i)$, you have the Gram matrix (and it’s easy to see that it will be positive semi-definite). *Why?*

  – Indirect: If you have the Kernel, write down the Kernel matrix $K_{ij}$, and show that it is a legitimate kernel, without an explicit construction of $\phi(x_i)$
Mercer’s condition

Let \( K(\mathbf{x}, \mathbf{z}) \) be a function that maps two \( n \) dimensional vectors to a real number

\[ K \text{ is a valid kernel if for every finite set } \{x_1, x_2, \cdots \}, \text{ for any choice of real valued } c_1, c_2, \cdots, \text{ we have } \]

\[ \sum_i \sum_j c_i c_j K(x_i, x_j) \geq 0 \]
Polynomial kernels

• Linear kernel: \( k(x, z) = x^Tz \)

• Polynomial kernel of degree \( d \): \( k(x, z) = (x^Tz)^d \)
  – only \( d \)th-order interactions

• Polynomial kernel up to degree \( d \): \( k(x, z) = (x^Tz + c)^d \) \((c > 0)\)
  – all interactions of order \( d \) or lower
Gaussian Kernel
(or the radial basis function kernel)

\[ K_{rbf}(x, z) = \exp \left( -\frac{\|x - z\|^2}{c} \right) \]

- \((x - z)^2\): squared Euclidean distance between \(x\) and \(z\)
- \(c = \sigma^2\): a free parameter
- very small \(c\): \(K \approx\) identity matrix (every item is different)
- very large \(c\): \(K \approx\) unit matrix (all items are the same)

- \(k(x, z) \approx 1\) when \(x, z\) close
- \(k(x, z) \approx 0\) when \(x, z\) dissimilar
Gaussian Kernel
(or the radial basis function kernel)

\[ K_{rbf}(\mathbf{x}, \mathbf{z}) = \exp \left( -\frac{||\mathbf{x} - \mathbf{z}||^2}{c} \right) \]

- \((\mathbf{x} - \mathbf{z})^2\): squared Euclidean distance between \(\mathbf{x}\) and \(\mathbf{z}\)
- \(c = \sigma^2\): a free parameter
- very small \(c\): \(K \approx\) identity matrix (every item is different)
- very large \(c\): \(K \approx\) unit matrix (all items are the same)

- \(k(\mathbf{x}, \mathbf{z}) \approx 1\) when \(\mathbf{x}, \mathbf{z}\) close
- \(k(\mathbf{x}, \mathbf{z}) \approx 0\) when \(\mathbf{x}, \mathbf{z}\) dissimilar

Exercises:
1. Prove that this is a kernel.
2. What is the “blown up” feature space for this kernel?
Constructing New Kernels

You can construct new kernels $k'(x, x')$ from existing ones:

- Multiplying $k(x, x')$ by a constant $c$
  
  $$ck(x, x')$$

- Multiplying $k(x, x')$ by a function $f$ applied to $x$ and $x'$
  
  $$f(x)k(x, x')f(x')$$

- Applying a polynomial (with non-negative coefficients) to $k(x, x')$
  
  $$P( k(x, x') ) \text{ with } P(z) = \sum_i a_i z^i \text{ and } a_i \geq 0$$

- Exponentiating $k(x, x')$
  
  $$\exp(k(x, x'))$$
Constructing New Kernels (2)

• You can construct $k'(x, x')$ from $k_1(x, x')$, $k_2(x, x')$ by:
  
  – Adding $k_1(x, x')$ and $k_2(x, x')$:
    $$k_1(x, x') + k_2(x, x')$$

  – Multiplying $k_1(x, x')$ and $k_2(x, x')$:
    $$k_1(x, x')k_2(x, x')$$
Constructing New Kernels (2)

• You can construct $k'(x, x')$ from $k_1(x, x'), k_2(x, x')$ by:
  
  – Adding $k_1(x, x')$ and $k_2(x, x')$:
    $k_1(x, x') + k_2(x, x')$

  – Multiplying $k_1(x, x')$ and $k_2(x, x')$:
    $k_1(x, x')k_2(x, x')$

• Also:

  – If $\phi(x) \in \mathbb{R}^m$ and $k_m(z, z')$ a valid kernel in $\mathbb{R}^m$, $k(x, x') = k_m(\phi(x), \phi(x'))$ is also a valid kernel

  – If $A$ is a symmetric positive semi-definite matrix, $k(x, x') = xAx'$ is also a valid kernel
Kernel Trick: An example

Let the blown up feature space represent the space of all $3^n$ conjunctions. Then,

$$K(x, z) = \sum_i \phi_i(x)\phi_i(z) = 2^{\text{same}(x,z)}$$

where $\text{same}(x,z)$ is the number of features that have the same value for both $x$ and $z$. 
This lecture

✓ Support vectors
✓ Kernels
✓ The kernel trick
✓ Properties of kernels

5. Another example of the kernel trick
Kernel Trick: An example

Let the blown up feature space represent the space of all $3^n$ conjunctions. Then,

$$K(x, z) = \sum_{i} \phi_i(x)\phi_i(z) = 2^{same(x, z)}$$

where $same(x, z)$ is the number of features that have the same value for both $x$ and $z$

Example: Take $n=3$; $x=(001)$, $z=(011)$, we have conjunctions of size 0, 1, 2, 3
Kernel Trick: An example

Let the blown up feature space represent the space of all $3^n$ conjunctions. Then,

$$K(x, z) = \sum_{i} \phi_i(x)\phi_i(z) = 2^{same(x, z)}$$

where $same(x, z)$ is the number of features that have the same value for both $x$ and $z$

Example: Take $n=3$; $x=(001)$, $z=(011)$, we have conjunctions of size 0,1,2,3

Proof: let $m=same(x, z)$; construct “surviving” conjunctions by
Kernel Trick: An example

Let the blown up feature space represent the space of all $3^n$ conjunctions. Then,

$$K(x, z) = \sum_i \phi_i(x)\phi_i(z) = 2^{same(x, z)}$$

where $same(x, z)$ is the number of features that have the same value for both $x$ and $z$

Example: Take $n=3$; $x=(001)$, $z=(011)$, we have conjunctions of size 0,1,2,3

Proof: Let $m=same(x, z)$; construct “surviving” conjunctions by

1. choosing to include one of these $k$ literals with the right polarity in the conjunctions, or
Kernel Trick: An example

Let the blown up feature space represent the space of all $3^n$ conjunctions. Then,

$$K(x, z) = \sum_i \phi_i(x)\phi_i(z) = 2^{\text{same}(x,z)}$$

where $\text{same}(x,z)$ is the number of features that have the same value for both $x$ and $z$

Example: Take $n=3$; $x=(001)$, $z=(011)$, we have conjunctions of size 0,1,2,3

**Proof:** let $m=\text{same}(x, z)$; construct “surviving” conjunctions by

1. choosing to include one of these $k$ literals with the right polarity in the conjunctions, or
2. choosing to not include it at all.
Kernel Trick: An example

Let the blown up feature space represent the space of all $3^n$ conjunctions. Then,

$$K(x, z) = \sum_{i} \phi_i(x) \phi_i(z) = 2^{\text{same}(x,z)}$$

where $\text{same}(x,z)$ is the number of features that have the same value for both $x$ and $z$

Example: Take $n=3$; $x=(001)$, $z=(011)$, we have conjunctions of size 0,1,2,3

Proof: let $m=\text{same}(x, z)$; construct “surviving” conjunctions by

1. choosing to include one of these $k$ literals with the right polarity in the conjunctions, or
2. choosing to not include it at all.

Conjunctions with literals outside this set disappear.
Exercises

1. Show that this argument works for a specific example
   - Take $X=\{x_1, x_2, x_3, x_4\}$
   - $\phi(x)$ = The space of all $3^n$ conjunctions ; $|\phi(x)| = 81$
   - Consider $x=(1100), z=(1101)$
   - Write $\phi(x), \phi(z)$, the representation of $x, z$ in the $\phi$ space
   - Compute $\phi(x)^T \phi(z)$
   - Show that
     \[
     K(x,z) = \phi(x)^T \phi(z) = \sum_i \phi_i(z) \phi_i(x) = 2^{\text{same}(x,z)} = 8
     \]

2. Try to develop another kernel, e.g., where the space of all conjunctions of size 3 (exactly)
Summary: Kernel trick

- To make the final prediction, we are computing dot products.

- The kernel trick is a computational trick to compute dot products in higher dimensional spaces.

- This is applicable not just to SVMs. The same idea can be extended to Perceptron too: the Kernel Perceptron.

- **Important**: All the bounds we have seen (eg: Perceptron bound, etc) depend on the underlying dimensionality.
  - By moving to a higher dimensional space, we are incurring a penalty on sample complexity.