



Lecture 5

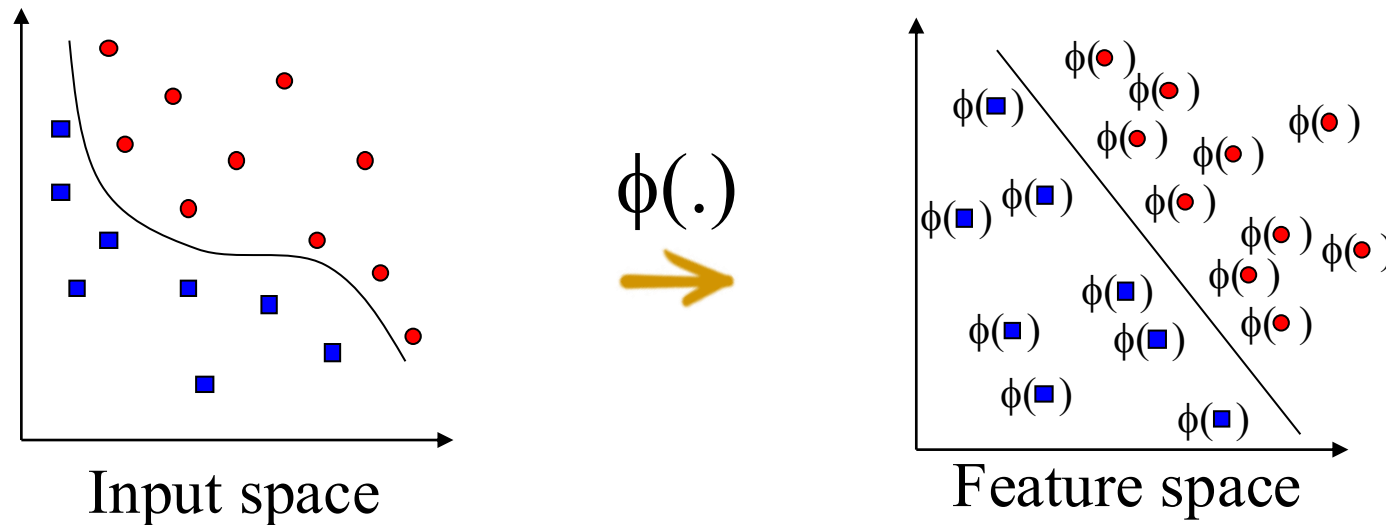
Kernels and SVM

Gabriele Pergola and Amina Asif
Machine Learning Algorithms and Practice
CS429

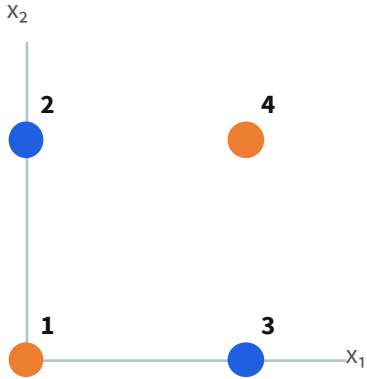
October 2026

Nonlinear Separation through Transformation

- Given a classification problem with a nonlinear boundary, we can, at times, find a mapping or transformation of the feature space which makes the classification problem linear separable in the transformed space



Examples: XOR Transformation



$$f(\mathbf{x};\boldsymbol{\theta}) = w_1x^{(1)} + w_2x^{(2)} + b = 0$$

$$(0,0): b < 0$$

$$(0,1): w_2 + b > 0$$

$$(1,0): w_1 + b > 0$$

$$(1,1): w_1 + w_2 + b < 0$$

$$f(\mathbf{x};\boldsymbol{\theta}) = w_1x^{(1)} + w_2x^{(2)} + w_3x^{(3)} + b = 0$$

$$(0,0,0): b < 0$$

$$(0,1,0): w_2 + b > 0$$

$$(1,0,0): w_1 + b > 0$$

$$(1,1,\sqrt{2}): w_1 + w_2 + \sqrt{2}w_3 + b < 0$$

$$w_1 = 2, \quad w_2 = 2, \quad w_3 = -3, \quad b = -1$$

$x^{(1)}$	$x^{(2)}$	y
0	0	-1
0	1	+1
1	0	+1
1	1	-1

$$\phi\left(\begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix}\right) = \begin{bmatrix} x^{(1)2} \\ x^{(2)2} \\ \sqrt{2}x^{(1)}x^{(2)} \end{bmatrix}$$

$x'^{(1)}$	$x'^{(2)}$	$x'^{(3)}$	y
0	0	0	-1
0	1	0	+1
1	0	0	+1
1	1	$\sqrt{2}$	-1

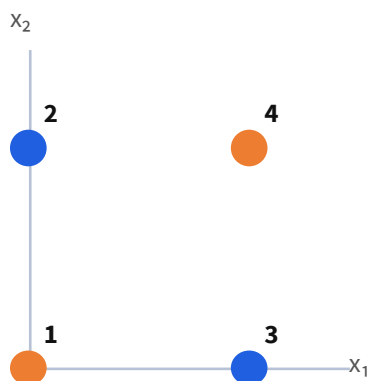
Solution

- Non-linear transformations of the features may allow us to perform effective partitioning

$$\phi(\mathbf{x}): \mathbf{R}^d \rightarrow \mathbf{R}^{d'}$$

$x^{(1)}$	$x^{(2)}$	y
0	0	-1
0	1	+1
1	0	+1
1	1	-1

No Solution



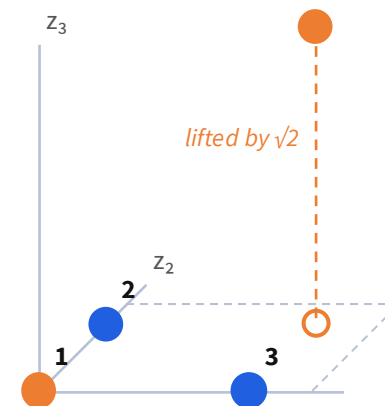
Transformation

$$\phi\left(\begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix}\right) = \begin{bmatrix} x^{(1)2} \\ x^{(2)2} \\ \sqrt{2}x^{(1)}x^{(2)} \end{bmatrix}$$

$$f(\mathbf{x}; \mathbf{w}) = w_1 x^{(1)} + w_2 x^{(2)} + w_3 x^{(3)} + b = 0$$

$x'(1)$	$x'(2)$	$x'(3)$	y
0	0	0	-1
0	1	0	+1
1	0	0	+1
1	1	$\sqrt{2}$	-1

Solution: $w_1 = 2, w_2 = 2, w_3 = -3, b = -1$



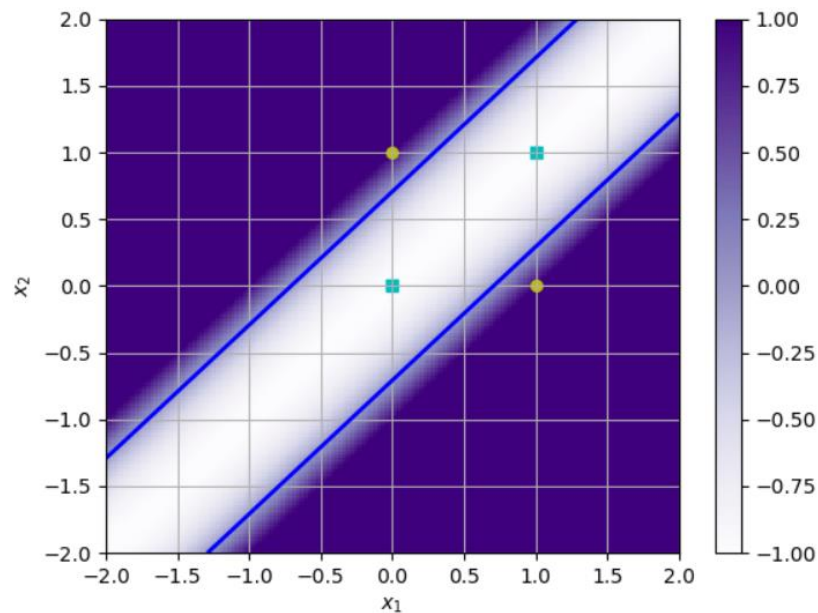
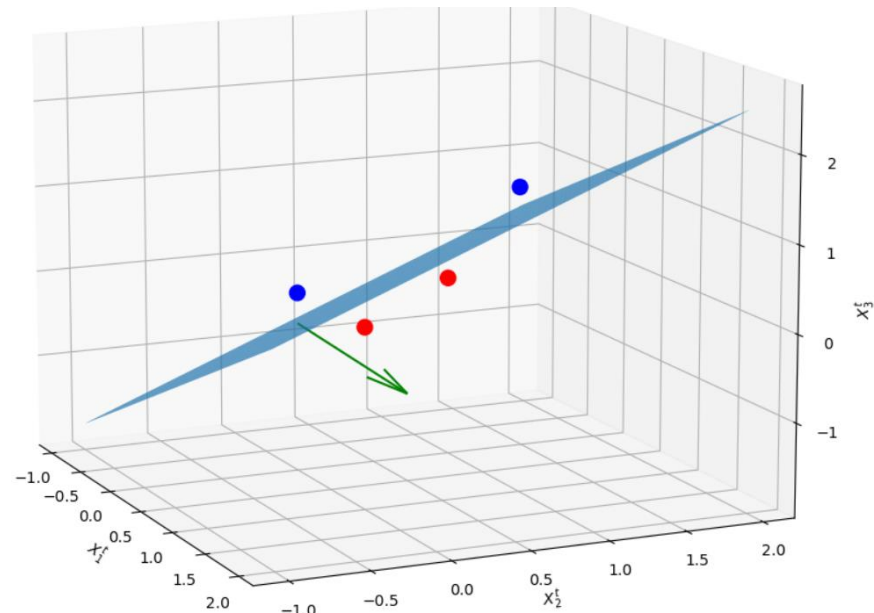
```
import numpy as np
from sklearn.svm import LinearSVC as Classifier
from plotit import *
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
```

```
X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([-1,1,1,-1])
clf2d = Classifier(C=1000).fit(X, y)
f = clf2d.predict(X)
print('Coefficients before Transformation:', clf2d.coef_,clf2d.intercept_)
print('Predictions before Transformation:',f)
```

```
transform = lambda x: np.hstack((x**2,np.atleast_2d(np.sqrt(2)*x[:,0]*x[:,1]).T))
Xt = transform(X)
print(Xt)
clf = Classifier(C=1000).fit(Xt, y)
f = clf.predict(Xt)
print('Coefficients after Transformation:',clf.coef_,clf.intercept_)
print('Predictions after Transformation:',f)
```

```
# showing the plane in 3d
xx,yy = np.arange(-1, 2, 0.01), np.arange(-1, 2, 0.01)
yy = xx, yy = np.meshgrid(xx, yy)
zz = -(clf.coef_[0,0]*xx+clf.coef_[0,1]*yy+clf.intercept_[0])/(clf.coef_[0,2])
fig = plt.figure(); ax = fig.add_subplot(111, projection='3d')
Xp,Xn = Xt[y==1,:],Xt[y!=1,:]
ax.scatter(Xp[:,0], Xp[:,1], Xp[:,2],color = 'red',alpha=1,s=100)
ax.scatter(Xn[:,0], Xn[:,1], Xn[:,2],color = 'blue',alpha=1,s=100)
ax.plot_surface(xx, yy, zz,linewidth=0, antialiased=True)
ax.set_xlabel('$X^1$'); ax.set_ylabel('$X^2$'); ax.set_zlabel('$X^3$')
# Normal vector (coef_ of the SVM)
normal_vector = clf.coef_[0]
start_point = [0, 0, -clf.intercept_[0] / clf.coef_[0,2]]
ax.quiver(start_point[0], start_point[1], start_point[2],
          normal_vector[0], normal_vector[1], normal_vector[2], length=1, color='green', normalize=True)

# showing the boundary in 2d
plt.figure(); plotit(X,y,clf = clf.decision_function,transform = transform,conts=[0],extent=[-2,+2,-2,+2])
```



Examples: Transformation

- Does this mapping ($\phi_3: \mathbb{R}^2 \rightarrow \mathbb{R}^3$) do it?

$$\bullet \phi \left(\begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix} \right) = \begin{bmatrix} x^{(1)2} \\ x^{(2)2} \\ \sqrt{2}x^{(1)}x^{(2)} \end{bmatrix}$$

- A nonlinear feature transformation does not necessarily increase dimensionality.

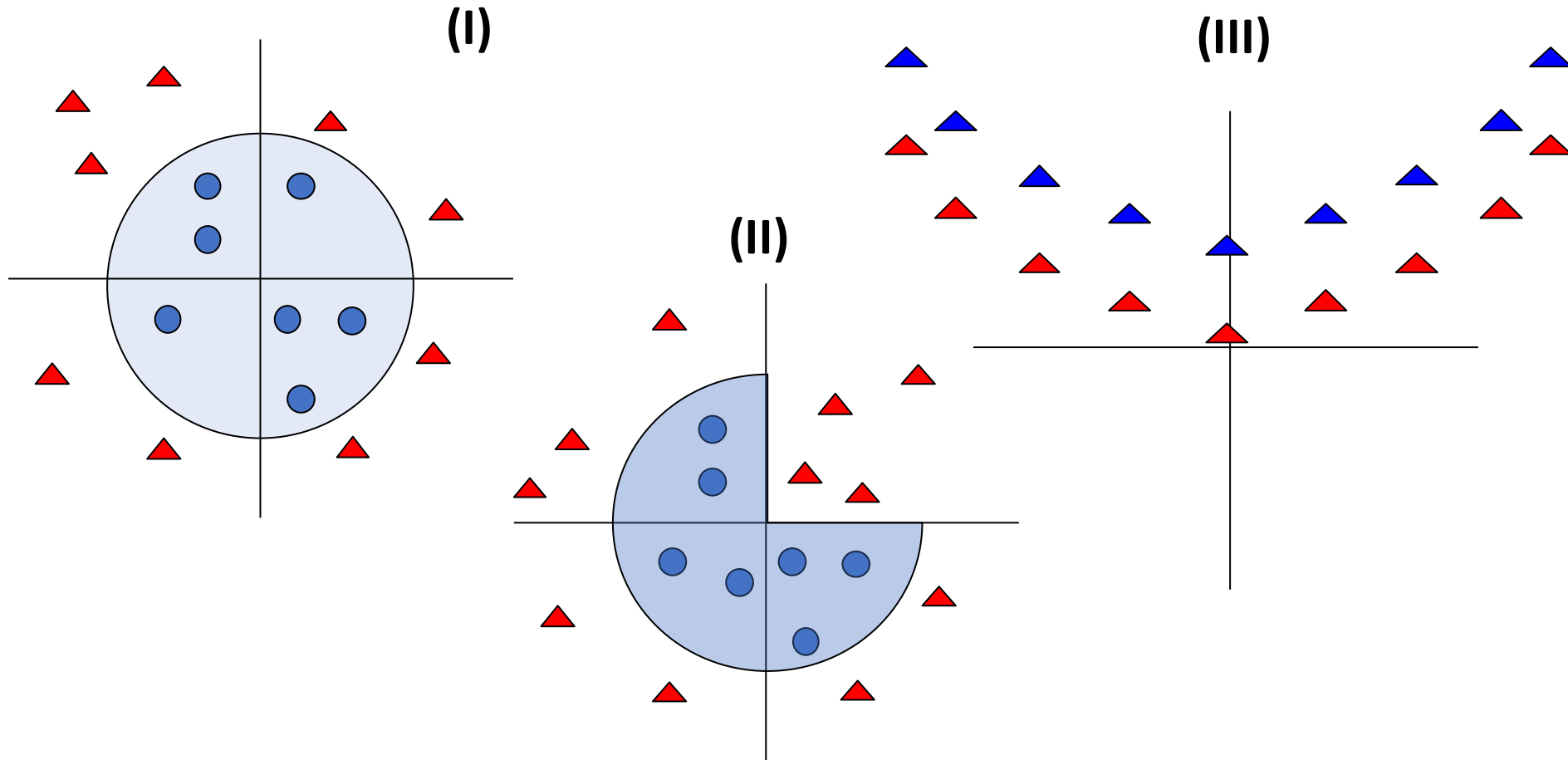
What about this one? $\boxed{\phi: \mathbb{R}^2 \rightarrow \mathbb{R}}$

$$\bullet \phi \left(\begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix} \right) = (x^{(1)} + x^{(2)} - 1)^2$$

- This is a nonlinear feature transformation from a **2-d** input space to a **1-d** feature space ($\boxed{\phi: \mathbb{R}^2 \rightarrow \mathbb{R}}$).
- Does this make XOR linearly separable?

Transformation Examples

- Can you find a transform that makes the following classification problems linear separable? Can you draw the data points in the new transformed feature space?



Solution

- Non-linear transformations of the features may allow us to perform effective partitioning

Have the **inner products** changed?
Have the **distances** changed?

How Does the Transformation Change the Inner Products?

- A **kernel matrix** records the inner product between every pair of examples.

Before: $\mathbf{K}(\mathbf{a}, \mathbf{b}) = \mathbf{a}^T \mathbf{b}$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \end{bmatrix}$$



After: $\mathbf{K}(\mathbf{a}, \mathbf{b}) = \boldsymbol{\phi}(\mathbf{a})^T \boldsymbol{\phi}(\mathbf{b})$

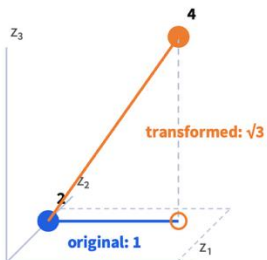
$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 4 \end{bmatrix}$$

Why does only one entry change?

Before: $(1, 1)^T (1, 1) = 2$

After: $(1, 1, \sqrt{2})^T (1, 1, \sqrt{2}) = 1 + 1 + 2 = 4$

Example 4 has acquired a new coordinate, so its squared length grows from 2 to 4. The other entries do not change, because the new coordinate is zero for examples 1, 2 and 3.



Original: $k(\mathbf{a}, \mathbf{b}) = \mathbf{a}^T \mathbf{b}$. Let's apply the transformation:

$$\boldsymbol{\phi}(\mathbf{a}) = \boldsymbol{\phi}\left(\begin{bmatrix} a^{(1)} \\ a^{(2)} \end{bmatrix}\right) = \begin{bmatrix} a^{(1)^2} \\ a^{(2)^2} \\ \sqrt{2}a^{(1)}a^{(2)} \end{bmatrix}$$

$$\boldsymbol{\phi}(\mathbf{b}) = \boldsymbol{\phi}\left(\begin{bmatrix} b^{(1)} \\ b^{(2)} \end{bmatrix}\right) = \begin{bmatrix} b^{(1)^2} \\ b^{(2)^2} \\ \sqrt{2}b^{(1)}b^{(2)} \end{bmatrix}$$

And compute the dot-product after transformation:

$$\begin{aligned} k_{\phi}(\mathbf{a}, \mathbf{b}) &= \boldsymbol{\phi}(\mathbf{a})^T \boldsymbol{\phi}(\mathbf{b}) \\ &= a^{(1)^2} b^{(1)^2} + a^{(2)^2} b^{(2)^2} + \sqrt{2}a^{(1)}a^{(2)}\sqrt{2}b^{(1)}b^{(2)} \\ &= (a^{(1)}b^{(1)})^2 + (a^{(2)}b^{(2)})^2 + 2a^{(1)}a^{(2)}b^{(1)}b^{(2)} \\ &= (a^{(1)}b^{(1)} + a^{(2)}b^{(2)})^2 = (\mathbf{a}^T \mathbf{b})^2 \end{aligned}$$

The above transformation is equivalent to the polynomial kernel

The inner products have changed. What has happened to the **distances**?

How Does the Transformation Change the Inner Products?

- A **kernel matrix** records the inner product between every pair of examples.

Before: $\mathbf{K}(\mathbf{a}, \mathbf{b}) = \mathbf{a}^\top \mathbf{b}$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \end{bmatrix}$$



After: $\mathbf{K}(\mathbf{a}, \mathbf{b}) = \boldsymbol{\phi}(\mathbf{a})^\top \boldsymbol{\phi}(\mathbf{b})$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 4 \end{bmatrix}$$

The **kernel trick**:

$$K_\phi(a, b) = (a^\top b)^2$$

So, from the original coordinates only:

$$K_\phi(x_4, x_4) = ((1, 1)^\top (1, 1))^2 = 2^2 = 4$$

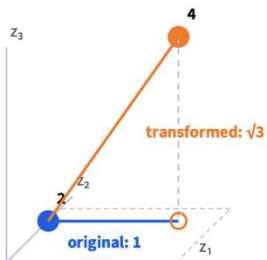
Why does only one entry change?

Before: $(1, 1)^\top (1, 1) = 2$

After: $(1, 1, \sqrt{2})^\top (1, 1, \sqrt{2}) = 1 + 1 + 2 = 4$

Example 4 has acquired a new coordinate, so its squared length grows from 2 to 4. The other entries do not change, because the new coordinate is zero for examples 1, 2 and 3.

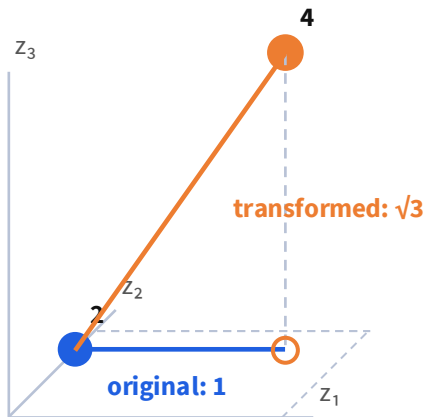
- With the kernel, **no three dimensional vector is needed.**



The inner products have changed. What has happened to the **distances**?

How Does the Transformation Change Distances?

The same two examples, $x_2 = (0, 1)$ and $x_4 = (1, 1)$, measured three ways.



Kernel matrix after the transformation

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 4 \end{bmatrix}$$

1 In the original space

$$d^2(x_2, x_4) = (0 - 1)^2 + (1 - 1)^2 = 1$$

2 In the transformed space, using $\phi(x_2) = (0, 1, 0)$ and $\phi(x_4) = (1, 1, \sqrt{2})$

$$d^2\phi(x_2, x_4) = (0 - 1)^2 + (1 - 1)^2 + (0 - \sqrt{2})^2 = 3$$

Point 4 has moved away from point 2: the squared distance grows from 1 to 3.

3 Without transforming the points?

$$d_\phi^2(a, b) = K(a, a) + K(b, b) - 2K(a, b)$$

Read from the kernel matrix: $K_{22} = 1$, $K_{44} = 4$, $K_{24} = 1 \Rightarrow 1 + 4 - 2(1) = 3$

The same answer from three kernel values.

Each can be computed from the original coordinates with $K(a, b) = (a^T b)^2$.

Worked Example: Distance Using Only Kernel Values

$x_2 = (0, 1)$ and $x_4 = (1, 1)$, original coordinates only. No transformed vector is calculated.

Linear kernel $K(a, b) = a^T b$

$$K(x_2, x_2) = 1$$

$$K(x_4, x_4) = 2$$

$$K(x_2, x_4) = 1$$

$$d^2 = 1 + 2 - 2(1) = 1$$

distance = 1

Quadratic kernel $K(a, b) = (a^T b)^2$

$$K(x_2, x_2) = 1$$

$$K(x_4, x_4) = 4$$

$$K(x_2, x_4) = 1$$

$$d^2 = 1 + 4 - 2(1) = 3$$

distance = $\sqrt{3}$

- The same two examples are further apart in the quadratic feature space.
- **By choosing a kernel, we implicitly choose a geometry for the data, and with it the margins the SVM can achieve.**

One Feature-Space Geometry, Three Ways to Describe It

Feature transformation

$$\varphi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

Kernel function

$$K(a, b) = \langle \varphi(a), \varphi(b) \rangle$$

Euclidean distance

$$d_\varphi^2 = K(a, a) + K(b, b) - 2K(a, b)$$



- **Coordinates, kernel values and distances describe one feature space geometry:** the transformation changes the representation of the data, and that representation determines different inner products and distances. And vice versa.
- The **kernel** lets us **work with those inner products directly**, without computing the transformed coordinates.

Feature Transformation $\leftrightarrow$ Kernel Function $\leftrightarrow$ Distance Function

$x^{(1)}$	$x^{(2)}$	y
0	0	-1
0	1	+1
1	0	+1
1	1	-1

$$k(a, b) = \mathbf{a}^T \mathbf{b}$$

i	1	2	3	4
1	0	0	0	0
2	0	1	0	1
3	0	0	1	1
4	0	1	1	2

$$d(a, b)$$

i	1	2	3	4
1	0	1	1	2
2	1	0	2	1
3	1	2	0	1
4	2	1	1	0

Pairwise dot product before transform

Pairwise distance before transform


 $\phi\left(\begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix}\right) = \begin{bmatrix} x^{(1)^2} \\ x^{(2)^2} \\ \sqrt{2}x^{(1)}x^{(2)} \end{bmatrix}$

Kernel (Gram) Matrix

Distance Matrix

$x'^{(1)}$	$x'^{(2)}$	$x'^{(3)}$	y
0	0	0	-1
0	1	0	+1
1	0	0	+1
1	1	$\sqrt{2}$	-1

Transformed Data

$$k_{\phi}(a, b) = \phi(\mathbf{a})^T \phi(\mathbf{b})$$

i	1	2	3	4
1	0	0	0	0
2	0	1	0	1
3	0	0	1	1
4	0	1	1	4

Pairwise dot product after transform

$$d_{\phi}(a, b) = \|\phi(\mathbf{a}) - \phi(\mathbf{b})\|^2$$

i	1	2	3	4
1	0	1	1	4
2	1	0	2	3
3	1	2	0	1
4	4	3	3	0

Pairwise distance after transform

Representation Learning

Kernel Based Learning

Metric Learning

Feature Transformation $\leftrightarrow$ Kernel Function $\leftrightarrow$ Distance Function

- So far, we have established that:
 - Whenever the features are transformed
 - The distance changes
 - The dot product values changes

Original: $k(\mathbf{a}, \mathbf{b}) = \mathbf{a}^T \mathbf{b}$. Let's apply the transformation:

$$\begin{aligned}\phi(\mathbf{a}) &= \phi\left(\begin{bmatrix} a^{(1)} \\ a^{(2)} \end{bmatrix}\right) = \begin{bmatrix} a^{(1)2} \\ a^{(2)2} \\ \sqrt{2}a^{(1)}a^{(2)} \end{bmatrix} \\ \phi(\mathbf{b}) &= \phi\left(\begin{bmatrix} b^{(1)} \\ b^{(2)} \end{bmatrix}\right) = \begin{bmatrix} b^{(1)2} \\ b^{(2)2} \\ \sqrt{2}b^{(1)}b^{(2)} \end{bmatrix}\end{aligned}$$

And compute the dot-product after transformation:

$$\begin{aligned}k_{\phi}(\mathbf{a}, \mathbf{b}) &= \phi(\mathbf{a})^T \phi(\mathbf{b}) \\ &= a^{(1)2}b^{(1)2} + a^{(2)2}b^{(2)2} + \sqrt{2}a^{(1)}a^{(2)}\sqrt{2}b^{(1)}b^{(2)} \\ &= (a^{(1)}b^{(1)})^2 + (a^{(2)}b^{(2)})^2 + 2a^{(1)}a^{(2)}b^{(1)}b^{(2)} \\ &= (a^{(1)}b^{(1)} + a^{(2)}b^{(2)})^2 = (\mathbf{a}^T \mathbf{b})^2\end{aligned}$$

The above transformation is equivalent to the polynomial kernel

- But it also means that
 - If I change the dot product, we will change the distance or in essence achieve an implicit transformation of the original feature space
 - **Kernel Trick:** Replace $\mathbf{a}^T \mathbf{b}$ with a Generalized Dot Product or kernel function $k(\mathbf{a}, \mathbf{b})$

Kernel SVM: Linking SVM Weights to Training Examples Representations

SVM

- The classifier must fit the training examples reasonably well, penalising margin violations.
- The classifier should avoid unnecessarily large weight magnitudes.

$$\min_{w,b} \frac{1}{2} \|w\|_2^2 + C \sum_{i=1}^N \ell_H(y_i f(x_i))$$

Keep weights small ($\frac{1}{2}\|w\|_2^2$)

Fit training examples (hinge loss)

We need to explicit the dot product in our optimisation function!

Could part of the learned weight vector be completely useless for classifying the training examples?

Intuition

The model learns from the training examples. Its weights determine the scores assigned to those examples.

Now imagine that a component of the weight vector points in a **direction that does not affect the score** of any training example.

What would happen if we removed it?

- The scores of all training examples would remain unchanged.
- The hinge-loss term would remain unchanged.
- But the weight norm would decrease.

Therefore, the regularised objective would improve.

Why should the SVM keep a weight component that does not improve training fit but still incurs a penalty? It should not.

All Features Vary, but One Direction Is Unnecessary

A very small training dataset and two classifiers

Example	x_1	x_2	Label
A	1	1	+1
B	-1	-1	-1

Classifier A

$$f_A(x) = x_1$$

$$w_A = (1, 0)$$

$$\|w_A\|^2 = 1$$

Classifier B

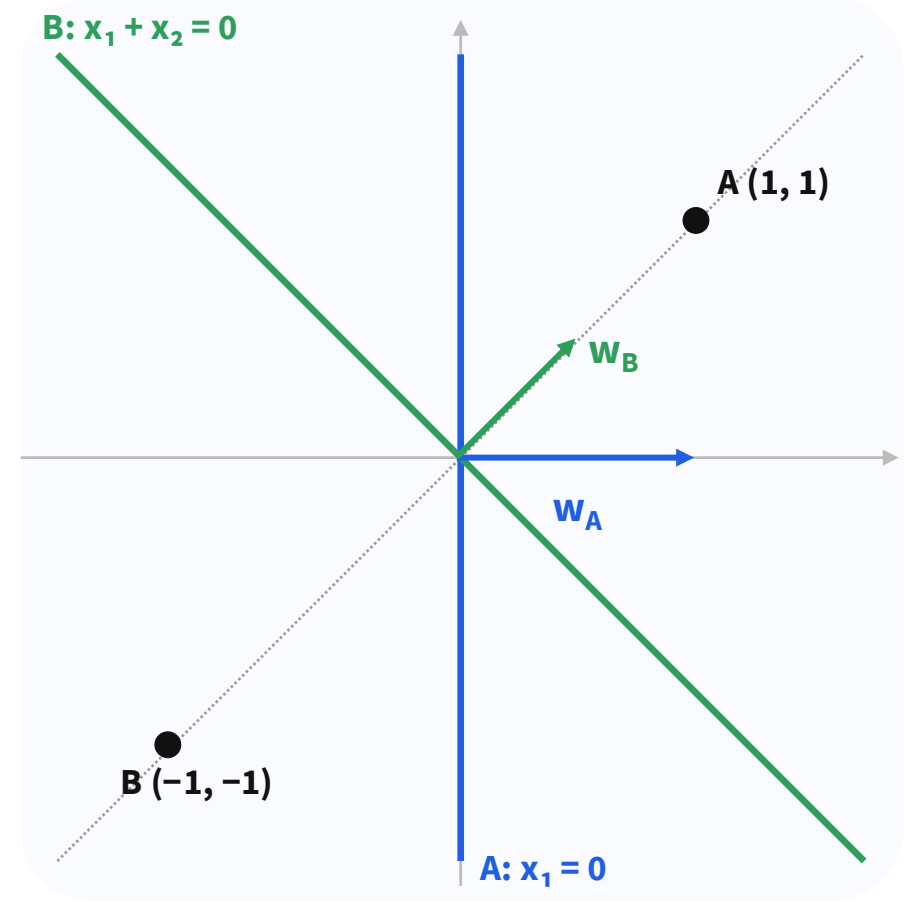
$$f_B(x) = \frac{x_1 + x_2}{2}$$

$$w_B = (0.5, 0.5)$$

$$\|w_B\|^2 = 0.5$$

What happens on the training examples?

	Classifier A	Classifier B
Score of A (1, 1)	+1	+1
Score of B (-1, -1)	-1	-1
Total hinge loss	0	0
Squared weight norm	1	0.5



- Both classifiers produce exactly the same scores on every training example, yet Classifier B has a smaller weight norm.
- Regularisation already guides the choice towards Classifier B (smaller $\|w_B\|^2$).
- **However, this time we notice the relation of $\|w_B\|$ with data...**

What is the difference between Classifier A and B?

Weights as Combination of the Training Data: Get rid of Unexplored Directions

- Classifier A's weight vector is $(1, 0)$, which points horizontally. We can **decompose** it into:

$$\begin{aligned} \mathbf{w}_A &= \mathbf{w}_{\parallel} + \mathbf{w}_{\perp} \\ (1, 0) &= (0.5, 0.5) + (0.5, -0.5) \end{aligned}$$

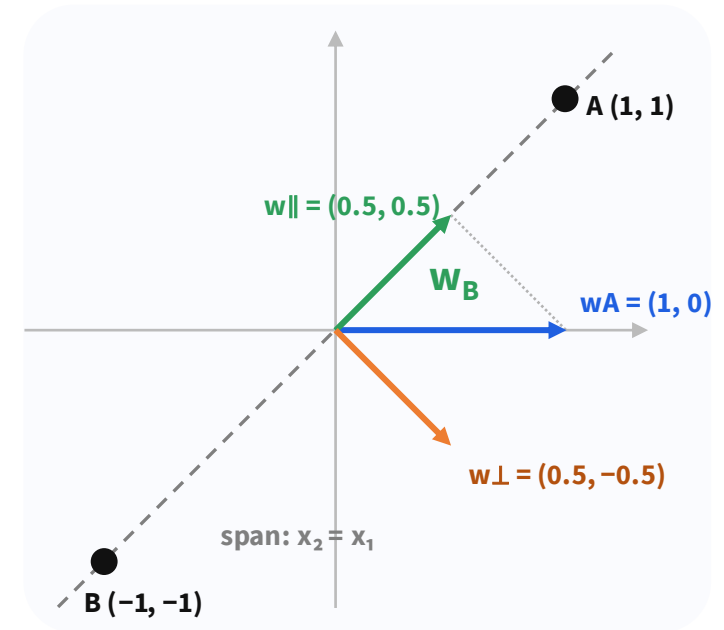
- $\mathbf{w}_{\parallel}$ points along the line generated by the training examples.
- $\mathbf{w}_{\perp}$ is perpendicular to that line.

And this is the important observation:

$$\underbrace{(0.5, -0.5)^T}_{\mathbf{w}_{\perp}} \underbrace{(1, 1)}_A = 0, \quad \underbrace{(0.5, -0.5)^T}_{\mathbf{w}_{\perp}} \underbrace{(-1, -1)}_B = 0$$

The perpendicular component has no effect on either training score (i.e., **same training error**, previous slide).

Yet it makes the original weight vector longer and **not preferable**.



By removing $\mathbf{w}_{\perp}$, we get precisely Classifier B:
 $\mathbf{w}_B = \mathbf{w}_{\parallel} = (0.5, 0.5)$



$\mathbf{w}_B$ lie on the dashed line/span (space) generated by the training data A and B (i.e., their linear combination).



Can we express the **optimal weight vector** as a linear combination of the training **examples**?

Representer Theorem of linear predictors

- We know that the discriminant function of the SVM can be written as:
- The [Representer theorem](#) (Scholkopf 2001) states that the weight vector of such a predictor is a linear combination of given training points, as anything orthogonal to the space of training points only adds cost to the optimization:

Thus:

Notice how the prediction function involves only dot-products.
Generalizing the dot product to a kernel function:

Now if I change the definition of the dot-product to a **generalized dot product** called a **kernel function**, I would be able to achieve transformations!

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$

$$\mathbf{w} = \alpha_1 \mathbf{x}_1 + \alpha_2 \mathbf{x}_2 + \dots + \alpha_N \mathbf{x}_N = \sum_{i=1}^N \alpha_i \mathbf{x}_i$$

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = \left(\sum_{i=1}^N \alpha_i \mathbf{x}_i \right)^T \mathbf{x} + b = \sum_{i=1}^N \alpha_i \mathbf{x}_i^T \mathbf{x} + b$$

$$k(\mathbf{a}, \mathbf{b}) = \mathbf{a}^T \mathbf{b}$$

e.g., Linear kernel

$$f(\mathbf{x}) = \sum_{i=1}^N \alpha_i k(\mathbf{x}_i, \mathbf{x}) + b$$

$$k(\mathbf{a}, \mathbf{b}) = \varphi(\mathbf{a})^T \varphi(\mathbf{b})$$

Representer Theorem of Linear Predictors

Representer Theorem Proof

Let's assume we are given training points $\{\mathbf{x}_j\}_{j=1}^N$ and an optimization objective

$$\min_{\mathbf{w}} R(\mathbf{w}) + L(X, Y; \mathbf{w})$$

where L (loss) only depends on $\mathbf{w}$ through inner (dot) products $\mathbf{w}^T \mathbf{x}$ and the regularizer $R(\mathbf{w})$ increases with $\|\mathbf{w}\|$ (e.g., $\frac{1}{2} \mathbf{w}^T \mathbf{w}$).

Define $\mathbf{w}_{\parallel} = \sum_{j=1}^N \alpha_j \mathbf{x}_j$ be a vector formed by an arbitrary linear combination of training points and $\mathbf{w}_{\perp}$ be a vector orthogonal to all such vectors.

Then, weight vector can be decomposed as: $\mathbf{w} = \mathbf{w}_{\parallel} + \mathbf{w}_{\perp}$.

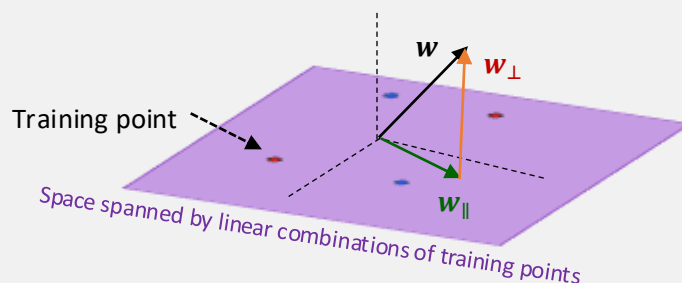
For any training point $\mathbf{x}_j$:

$$\mathbf{w}^T \mathbf{x}_j = (\mathbf{w}_{\parallel} + \mathbf{w}_{\perp})^T \mathbf{x}_j = \mathbf{w}_{\parallel}^T \mathbf{x}_j (\because \mathbf{w}_{\perp}^T \mathbf{x}_j = 0).$$

Hence, L is unchanged by $\mathbf{w}_{\perp}$.

However, as $\|\mathbf{w}\|^2 = \|\mathbf{w}_{\parallel}\|^2 + \|\mathbf{w}_{\perp}\|^2$ so any non-zero $\mathbf{w}_{\perp}$ will strictly increase $R(\mathbf{w})$. Therefore, the optimal $\mathbf{w}$ must have $\mathbf{w}_{\perp} = \mathbf{0}$, implying:

$$\mathbf{w} = \mathbf{w}_{\parallel} = \sum_{i=1}^N \alpha_i \mathbf{x}_i$$



The Representer Theorem: Expressing Weights Through Training Examples

Geometric illustration: the six-point example

For the six-point example, the hard-margin SVM has $w^* = (1, 0)$. One valid expansion uses the four margin-touching examples with equal coefficients of $1/4$:

$$w^* = \frac{1}{4}[(1, 1) + (1, -1) - (-1, 1) - (-1, -1)] = (1, 0)$$

This gives a concrete example of how the weight vector can be assembled from training observations.

Specialise to the SVM

$$w^* = \sum_{i=1}^N \alpha_i y_i x_i$$

where $\alpha_i \geq 0$ are coefficients obtained from the SVM optimisation problem.

Recall the support vectors:

- If $\alpha_i > 0$, the example is a support vector.
- If $\alpha_i = 0$, the example does not contribute to the weight expansion.

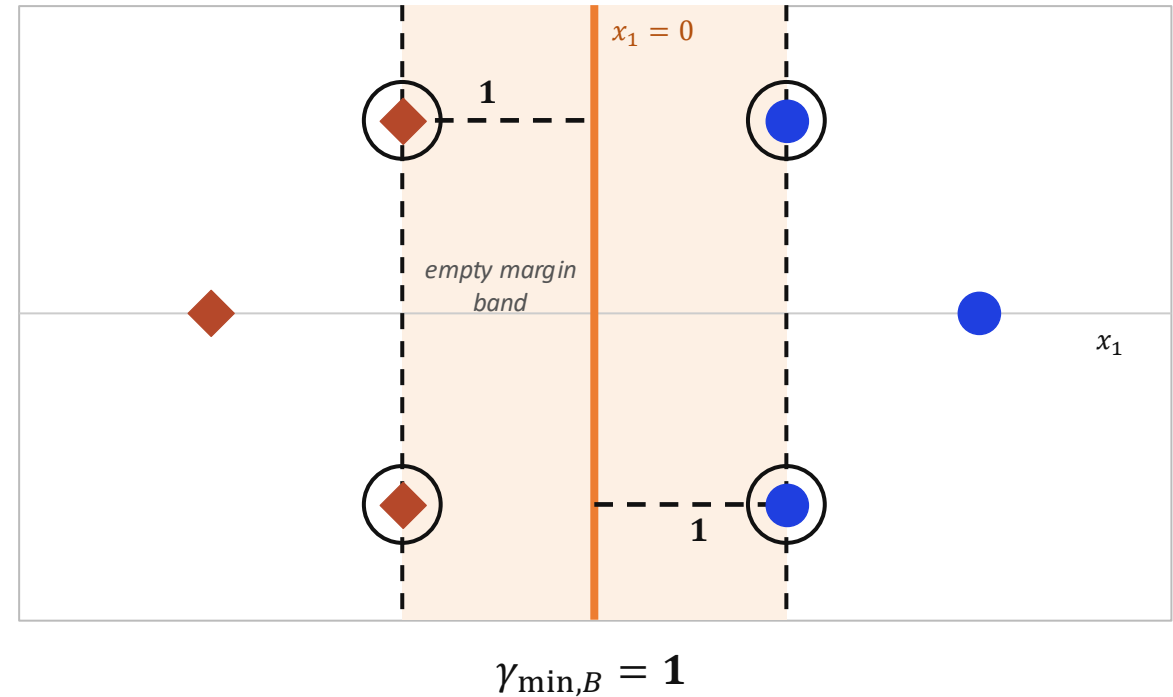
Therefore:

$$w^* = \sum_{i \in \text{SV}} \alpha_i y_i x_i$$

The learned SVM weight vector can be constructed from its support vectors.

- The general Representer Theorem explains why the solution lies in the span of training examples.

Maximum-margin separator $f_B(x) = x_1$



- Regularisation gives the optimal weight vector a special structure: it can be represented as a weighted combination of training examples.

Other Kernels

- We can change the definition of dot products to (almost) any other function
 - Each kernel will have its own underlying feature representation **even when an explicit feature representation does not exist**

Kernel Name	Kernel Function $K(\mathbf{a}, \mathbf{b})$
Linear	$\mathbf{a}^T \mathbf{b}$
Polynomial	$(\mathbf{a}^T \mathbf{b} + h)^d$
Radial Basis Function (RBF)	$\exp(-\gamma \ \mathbf{a} - \mathbf{b}\ ^2)$
Exponential RBF	$\exp(-\gamma \ \mathbf{a} - \mathbf{b}\)$
Sigmoid	$\tanh(\gamma(\mathbf{a}^T \mathbf{b}) + h)$
String kernels	Percentage of matching characters in $\mathbf{a}, \mathbf{b}$ $k(\text{"GATTACA"}, \text{"GATTACC"}) = 6/7$
Graph kernels	Number of shared sub-trees between graph representations of two drugs (Weisfeiler-Lehman (WL) subtree kernel)

Requirements for being a kernel

Any function k can be a kernel if its pairwise kernel or 'Gram' matrix

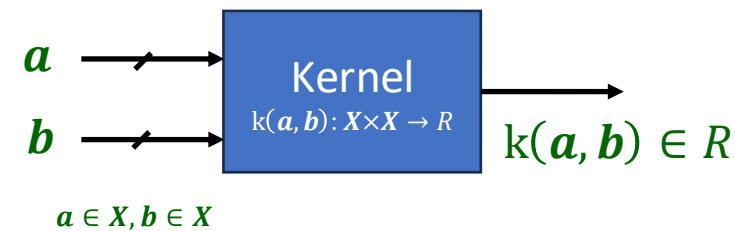
$$K = \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & k(x_1, x_3) \\ k(x_2, x_1) & k(x_2, x_2) & k(x_2, x_3) \\ k(x_3, x_1) & k(x_3, x_2) & k(x_3, x_3) \end{bmatrix}$$

is symmetric, positive semi-definite (for all given data).

And for any valid kernel, a corresponding transformation

$\mathbf{x} \rightarrow \phi(\mathbf{x})$ exists such that $k(\mathbf{a}, \mathbf{b}) = \phi(\mathbf{a})^T \phi(\mathbf{b})$.

Formally: [Moore-Aronszajn theorem](#)



Vector ($\mathbf{a} \in \mathbb{R}^d, \mathbf{b} \in \mathbb{R}^d$) or
unstructured example pairs
(e.g., strings or graphs)

Kernel SVM: Evaluation

Classifier

$$f(\mathbf{x}) = b + \mathbf{w}^T \mathbf{x} = b + \sum_{j=1}^N \alpha_j k(\mathbf{x}_j, \mathbf{x})$$

$$\mathbf{w} = \sum_{i=1}^N \alpha_i \mathbf{x}_i \text{ (Representer Theorem)}$$

$$\mathbf{w}^T \mathbf{w} = \left(\sum_{i=1}^N \alpha_i \mathbf{x}_i \right)^T \sum_{j=1}^N \alpha_j \mathbf{x}_j = \sum_{i,j=1}^N \alpha_i \alpha_j (\mathbf{x}_i^T \mathbf{x}_j)$$

Linear SVM

$$\min_{\mathbf{w}} \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{C}{N} \sum_{i=1}^N \max\{0, 1 - y_i f(\mathbf{x}_i; \mathbf{w})\}$$

$$\mathbf{x}_i^T \mathbf{x}_j \rightarrow k(\mathbf{x}_i, \mathbf{x}_j)$$

Kernel Trick

Kernelized SVM

$$\min_{\alpha, b} \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) + \frac{C}{N} \sum_{i=1}^N \max\left\{0, 1 - y_i \left(b + \sum_{j=1}^N \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) \right)\right\}$$

$$\max\left\{0, 1 - y_i \left(b + \sum_{j=1}^N \alpha_j \mathbf{x}_i^T \mathbf{x}_j \right)\right\}$$

Kernel SVM: Optimization with GD

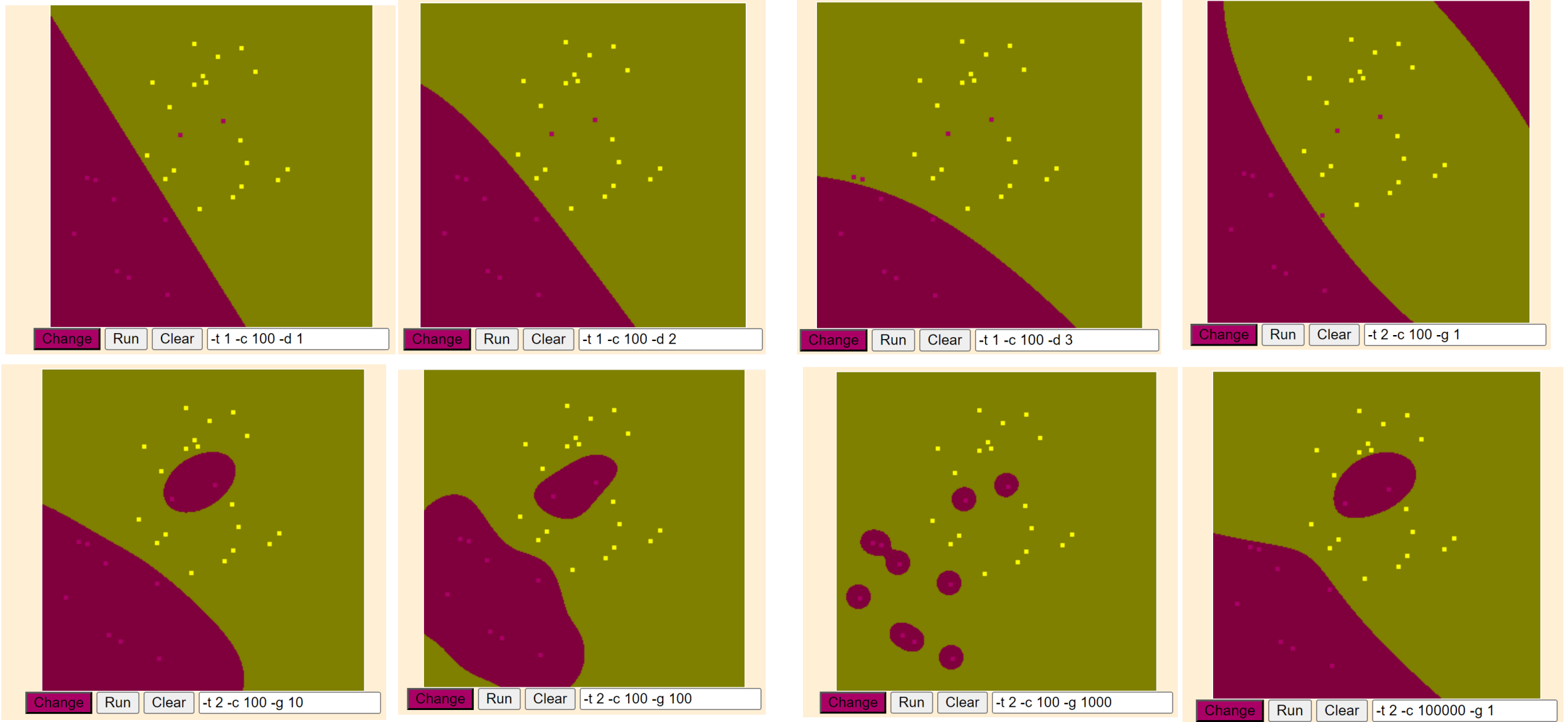
$$\min_{\alpha, b} D(\alpha, b) = \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) + \frac{C}{N} \sum_{i=1}^N \max \left\{ 0, 1 - y_i \left(b + \sum_{j=1}^N \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) \right) \right\}$$

$$\nabla_{\alpha_i} D = \sum_{j=1}^N \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) + -\frac{C}{N} \sum_{j=1}^N \begin{cases} y_j k(\mathbf{x}_i, \mathbf{x}_j) & \text{if } 1 - y_i \left(b + \sum_{j=1}^N \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) \right) > 0 \\ 0 & \text{else} \end{cases}$$

$$\nabla_b D = -\frac{C}{N} \sum_{j=1}^N \begin{cases} y_j & \text{if } 1 - y_i \left(b + \sum_{j=1}^N \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) \right) > 0 \\ 0 & \text{else} \end{cases}$$

$$\alpha^{(m)} \leftarrow \alpha^{(m-1)} - \eta \nabla D_{\alpha}(\alpha^{(k-1)})$$

- Use the Applet at: <https://www.csie.ntu.edu.tw/~cjlin/libsvm/>
- Study the impact of changing kernel type, kernel hyperparameters and C



Example: Solution of the OR problem

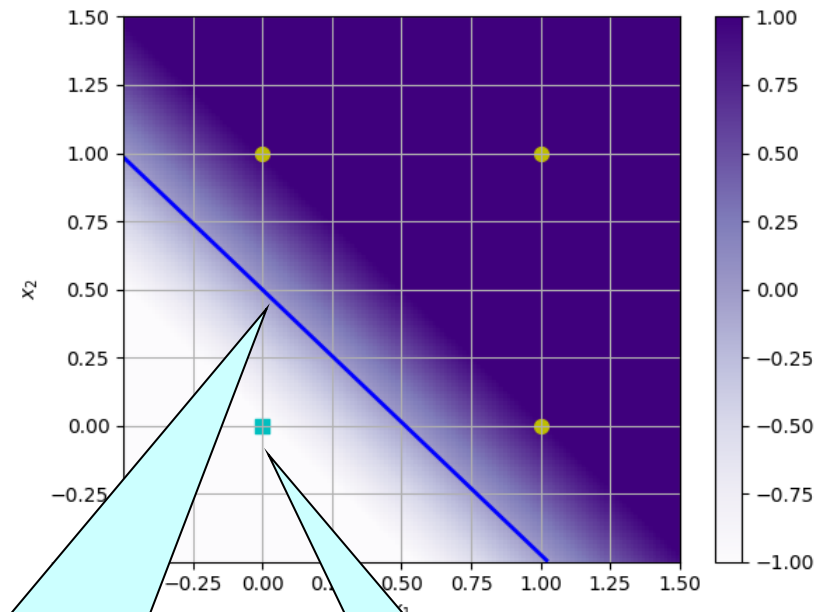
```
import numpy as np
from sklearn.svm import SVC as Classifier
```

```
X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([-1,1,1,1])
clf = Classifier(kernel = 'poly', degree = 1, C = 10).fit(X, y)
plotit(X,y,clf = clf.decision_function,conts=[0],extent=[-2,+2,-2,+2])
print("Alpha: ",clf.dual_coef_)
print(clf.support_vectors_)
print(clf.intercept_)
```

$$\mathbf{x}^i = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$
$$y = -1 \quad +1 \quad +1 \quad +1$$
$$\alpha_i = -2 \quad +1 \quad +1 \quad 0$$

$$\mathbf{w}^* = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$
$$b^* = -1$$

$$k(a, b) = a^T b$$



Optimal separating boundary

One of the 3 SVs

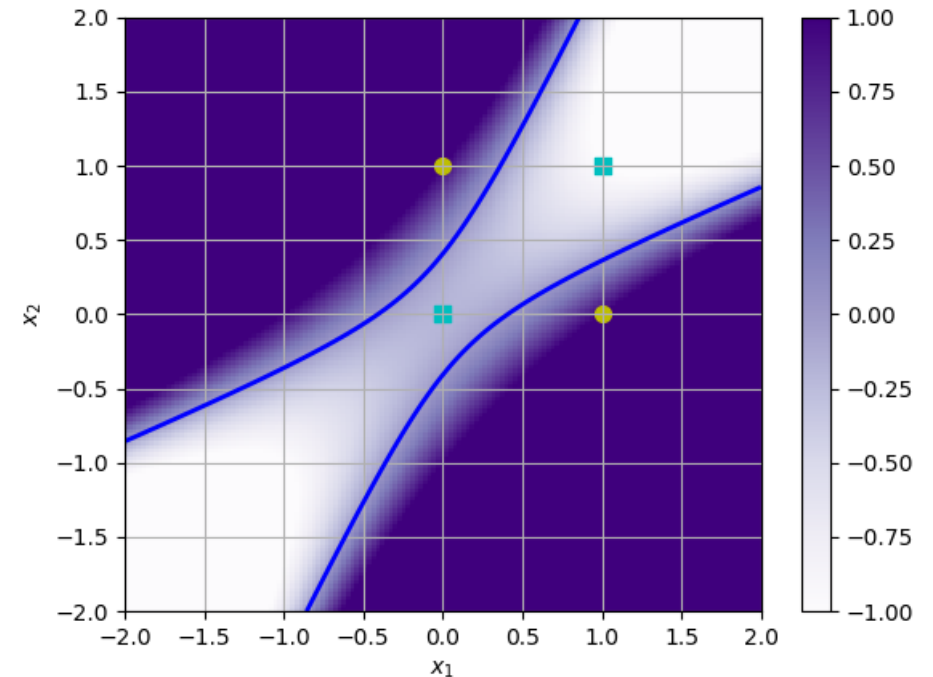
XOR

```
import numpy as np
from sklearn.svm import SVC as Classifier
```

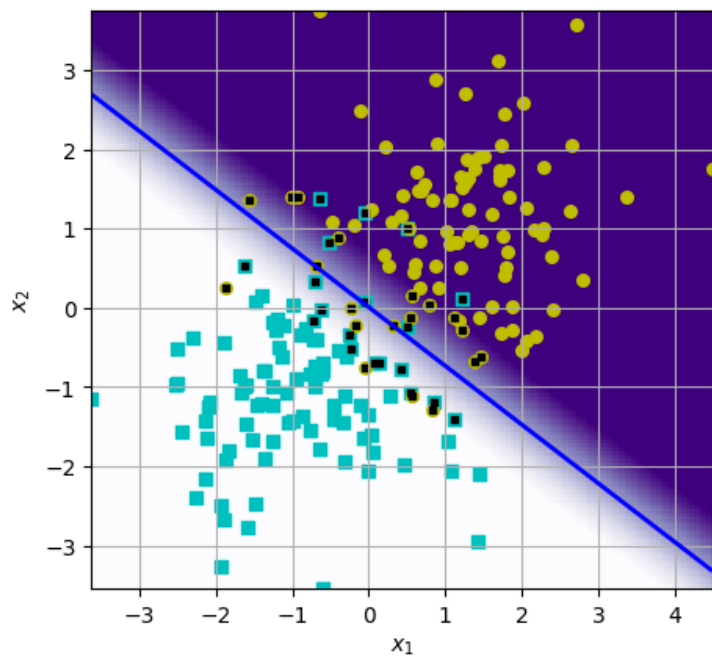
```
X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([-1,1,1,-1])
clf = Classifier(kernel = 'poly', degree = 2, C = 1).fit(X, y)
plotit(X,y,clf = clf.decision_function,conts=[0],extent=[-2,+2,-2,+2])
print("Alpha: ",clf.dual_coef_)
print(clf.support_vectors_)
print(clf.intercept_)
```

$$\mathbf{x}^i = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$
$$y = -1 \quad +1 \quad +1 \quad -1$$
$$\alpha_i = -1 \quad +0.7 \quad +0.7 \quad -0.4$$
$$b^* = -1$$

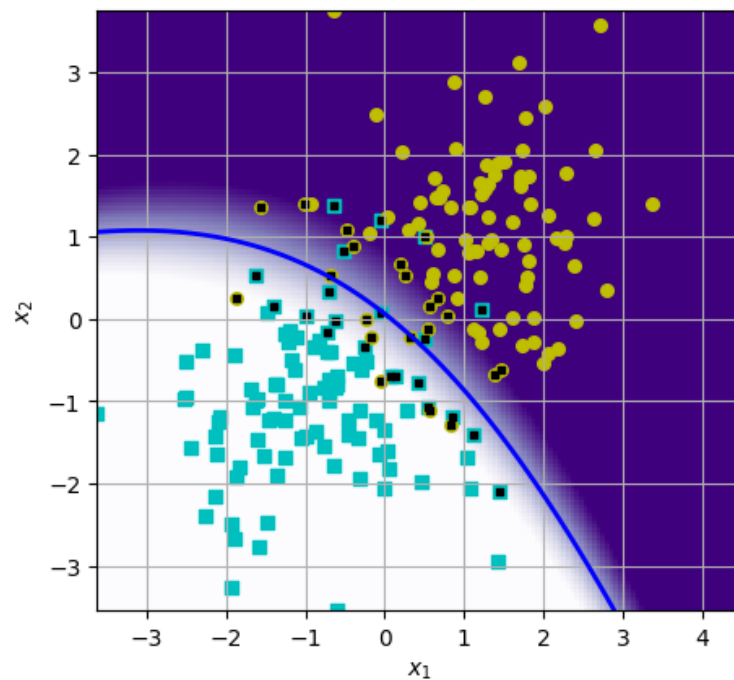
$$k(a, b) = (a^T b)^2$$



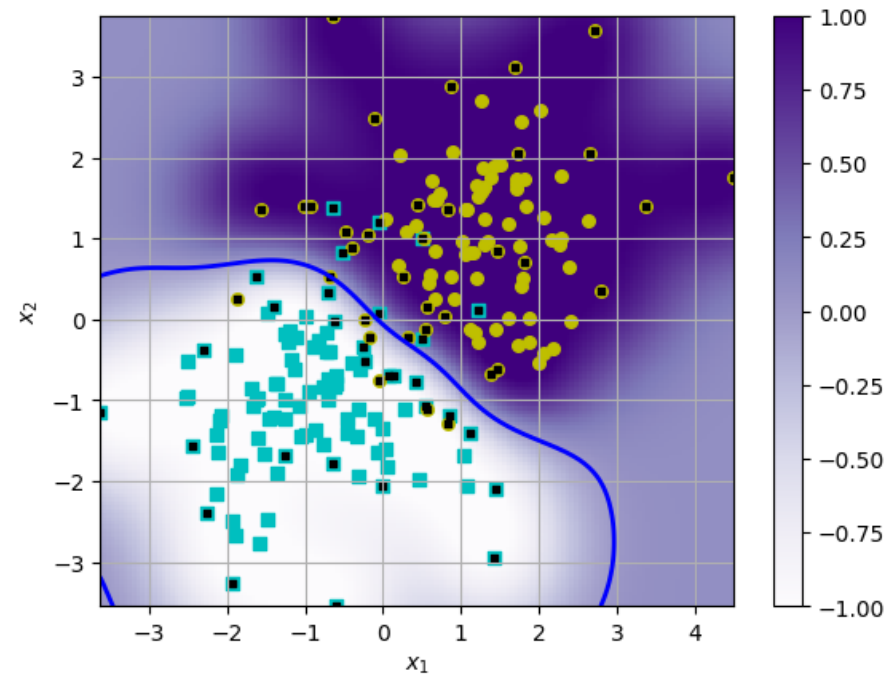
SVM with linear Kernel



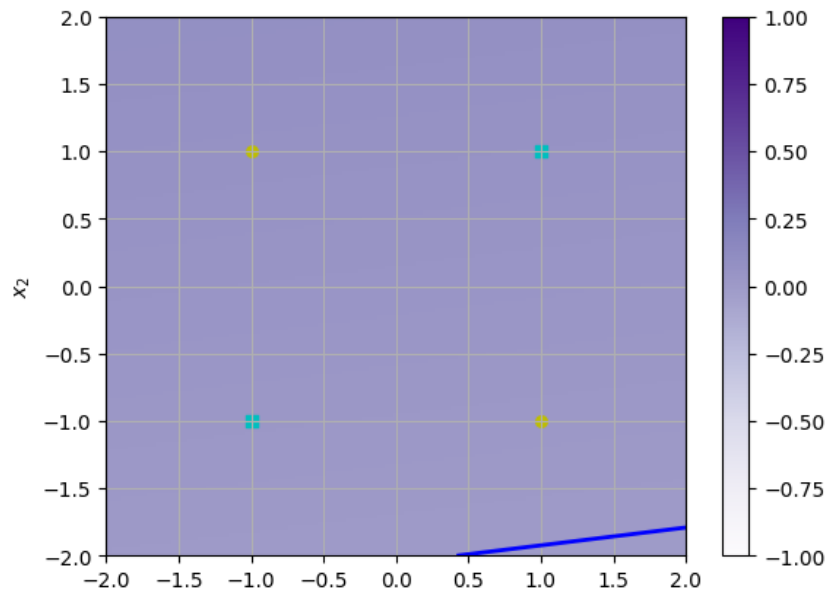
SVM with polynomial Kernel



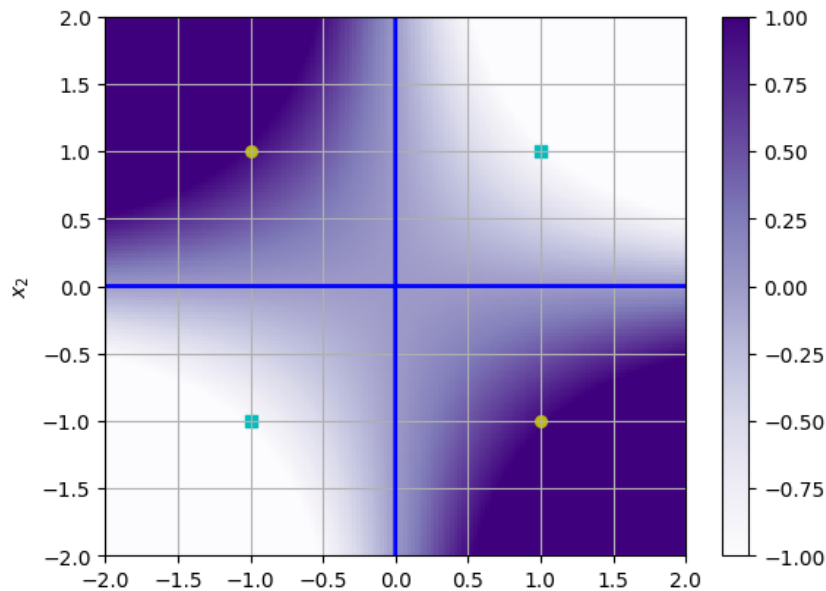
SVM with RBF Kernel



Linear SVM without Transform

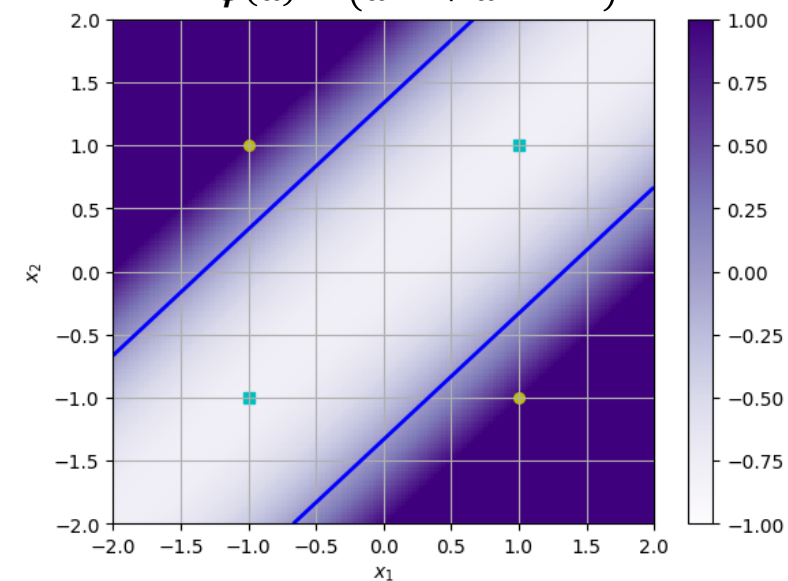


Linear SVM with transform
 $\phi(\mathbf{u}) = [u^{(1)^2} \quad u^{(2)^2} \quad \sqrt{2}u^{(1)}u^{(2)}]^T$

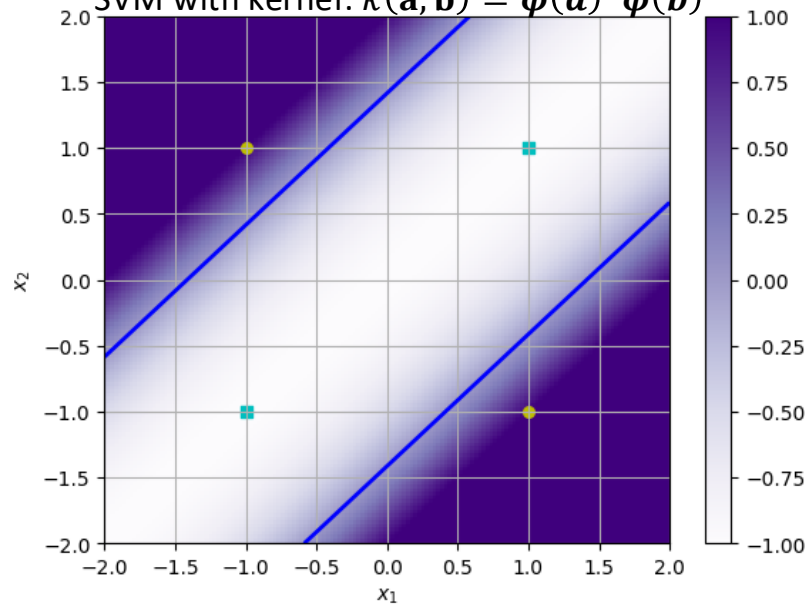


Linear SVM with transform

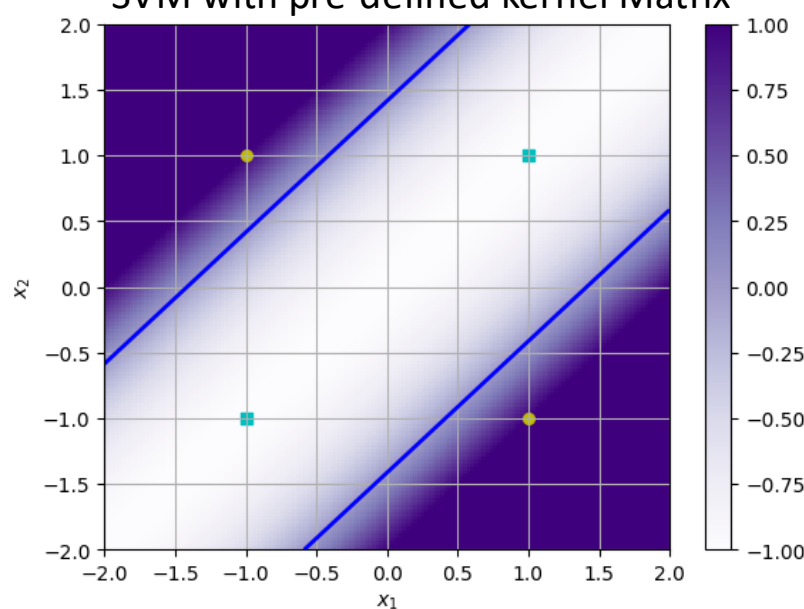
$$\phi(\mathbf{u}) = (\mathbf{u}^{(1)} + \mathbf{u}^{(2)} - 1)^2$$



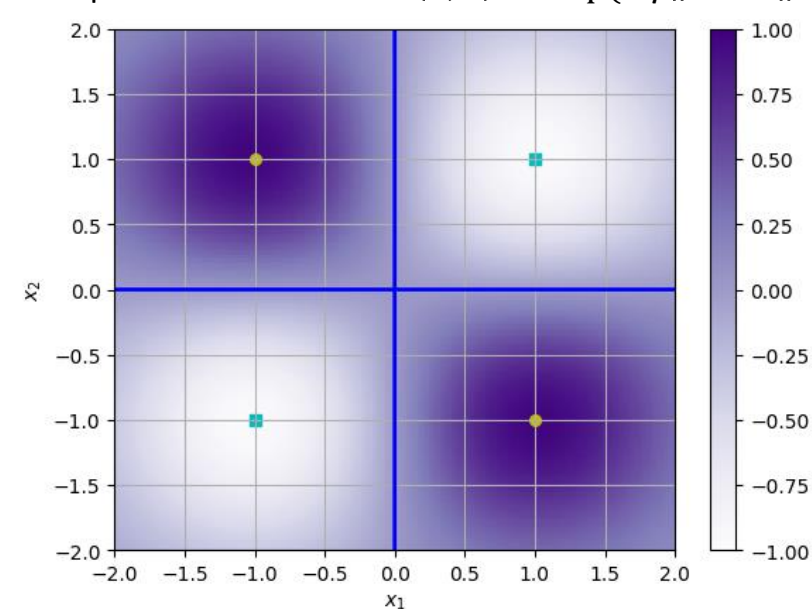
SVM with kernel: $k(\mathbf{a}, \mathbf{b}) = \phi(\mathbf{a})^T \phi(\mathbf{b})$



SVM with pre-defined kernel Matrix



SVM with pre-defined kernel: $k(\mathbf{a}, \mathbf{b}) = \exp(-\gamma \|\mathbf{a} - \mathbf{b}\|^2)$



Using the SVM

- Read:
- Ben-Hur, Asa, and Jason Weston. 2010. “A User’s Guide to Support Vector Machines.” In *Data Mining Techniques for the Life Sciences*, edited by Oliviero Carugo and Frank Eisenhaber, 223–39. Methods in Molecular Biology 609. Humana Press. http://dx.doi.org/10.1007/978-1-60327-241-4_13
- <http://pymml.sourceforge.net/doc/howto.pdf>

Handling data imbalance

- If the data is imbalanced (too much of one class and only a small number of examples from the other)
 - You can set an individual C for each class (called class weighting) or even per-example weighting
 - Can also be used to reflect a priori knowledge

$$\min_{\alpha, b} \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) + \frac{C}{N} \sum_{i=1}^N \max \left\{ 0, 1 - y_i \left(b + \sum_{j=1}^N \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) \right) \right\}$$

$$\min_{\alpha, b} \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i=1}^N c_i \max \left\{ 0, 1 - y_i \left(b + \sum_{j=1}^N \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) \right) \right\}$$

per-example weighting

Coding: https://scikit-learn.org/stable/auto_examples/svm/plot_weighted_samples.html

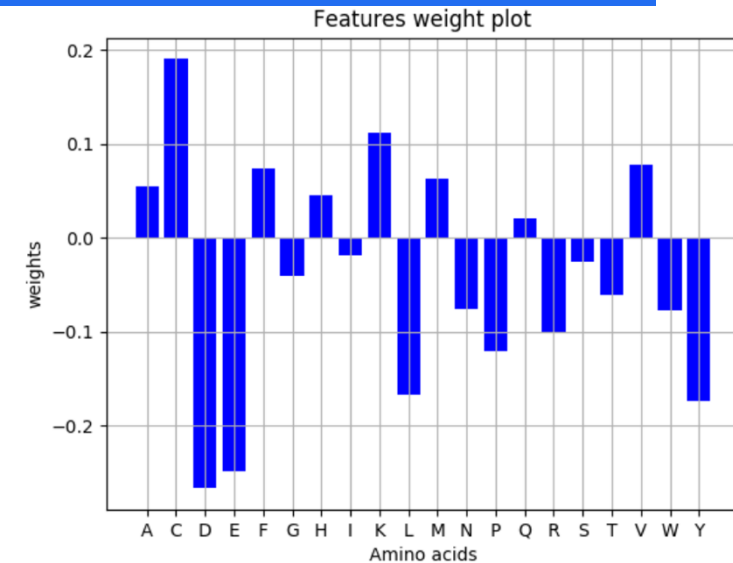
Strengths and Weaknesses of SVM

• Strengths

- Only a few training points (Support Vectors) determine the final boundary
- Very useful is the amount of training data is small (esp. in biomedical domains)
- Margin maximization and kernelized
- Optimization is relatively easy: No local optimal, unlike in neural networks
- It scales well to high dimensional data
- Tradeoff between classifier complexity and error can be controlled explicitly (through C)
- Non-traditional data like strings and graphs can be used as input to SVM, instead of feature vector: as the SVM only requires defining a kernel or degree of similarity between examples
- Completely interpretable and explainable
 - When using linear SVMs, the weight vector gives a clear indication of which features are important (if input data is appropriately scaled): $f(\mathbf{x}; \mathbf{w}) = w_1 x^{(1)} + w_2 x^{(2)} + \dots + w_d x^{(d)} + b$
 - When using non-linear SVMs, the decision can still be explained in terms of the degree of similarity to different training examples and coefficients: $f(\mathbf{x}; \alpha) = \alpha_1 k(\mathbf{x}, \mathbf{x}_1) + \alpha_2 \alpha_1 k(\mathbf{x}, \mathbf{x}_2) + \dots + \alpha_N \alpha_1 k(\mathbf{x}, \mathbf{x}_N) + b$

• Weaknesses

- Need to choose a “good” kernel function.
- Can be sensitive to the norm of data and thus normalization and standardization
 - Solution See: <https://scikit-learn.org/stable/modules/preprocessing.html>
- Large scale data
 - Solution: Kernel Approximation Algorithms (e.g., Nyström Approximation which represents a full NxN kernel on the basis of a much smaller mxm kernel over selected points as: $K \approx K_{Nm} K_{mm}^{-1} K_{Nm}^T$) or using randomized projections.



What are the underlying characteristics of an antimicrobial peptide? We can infer the relative importance of different amino acids in an antimicrobial peptide using the weights of the SVM.

Gull, Sadaf, Nauman Shamim, and Fayyaz Minhas. “AMAP: Hierarchical Multi-Label Prediction of Biologically Active and Antimicrobial Peptides.” *Computers in Biology and Medicine* 107 (April 1, 2019): 172–81.
<https://doi.org/10.1016/j.compbiomed.2019.02.018>.