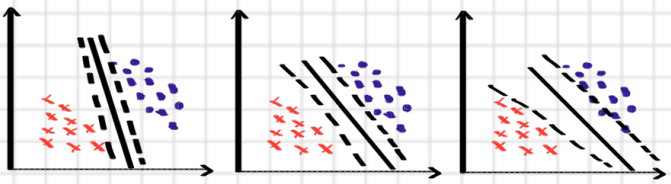
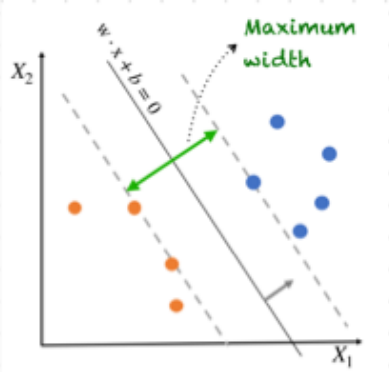
Lecture 14 - 31/10/24

Support Vector Machine (SVM)

- It is a **supervised learning method**
- Can be used both for:
  - Classification
  - Regression
- Intuition:
  - What's the **best decision boundary** for these training data points?



- If classes are **linearly separable**:
  - Best decision boundary → **line (hyperplane)**
  - The one with **maximum margin** from closest samples of both classes!



We recall here: **Hard margin SVM**

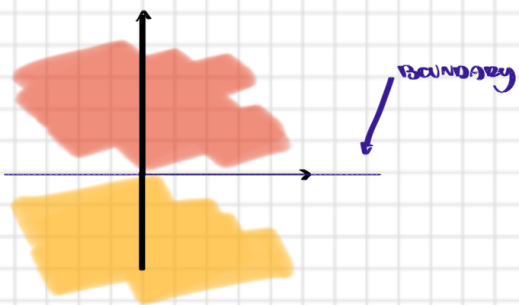
The "street" that forms inside is called **widest street** and the name induces the **underlined meaning**

- **Originally** SVM is used for **binary classification**
- **Combining several SVM's for multi-class** is possible though
- **Linear classifier:**

$y(x) = w \cdot x + b$, decision boundary : $y(x) = 0$

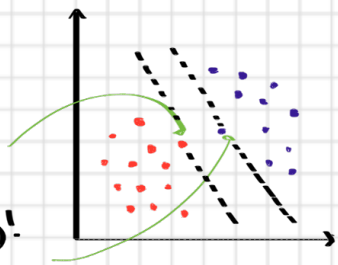
if $y(x_i) \geq 0$ then sample i is in class **+1**

if $y(x_i) < 0$ then sample i is in class **-1**



Hard Margin

- **Think** about which training **samples are relevant** for **widest street**?
  - The training samples that are relevant are the only ones that **contribute** to the decision boundary.
    - ↳ **Support Vectors!**
  - SVM **finds** support vectors **during training** & uses them to classify!



* LINEAR CLASSIFIER:

$y(x) = w \cdot x + b$

$y(x) = \sum_{i=1}^{n} \alpha_i y_i (x_i \cdot x) + b$

- n = # OF TRAINING SAMPLES!
- x_i = FEATURES VECTOR & OUTPUT OF TRAINING SAMPLE i
- x = INPUT VECTOR TO BE CLASSIFIED!
- $(x_i \cdot x)$ → DOT PRODUCT BETWEEN TRAINING DATAPOINTS & NEW DATAPOINT
- $\alpha_i \geq 0$

SOME SAMPLES MAY HAVE $\alpha_i = 0$. THE ONES WITH $\alpha_i > 0$ ARE SUPPORT VECTORS

TRAINING AN SVM MEANS FINDING VALUES FOR ALL THE α_i'S, EACH ASSOCIATED WITH A TRAINING SAMPLE SUCH THAT THE WIDTH OF THE STREET SEPARATING THE TWO CLASSES IS MAXIMIZED.
THIS MEANS FIGURING OUT WHICH $\alpha_i = 0$, IDENTIFYING SUP. VECTORS.
AFTER TRAINING KEEP ONLY SUP. VEC. ASSOCIATED WITH α_i FOR PREDICTION

- SO → TRAIN THE MODEL = FIND THE α_i MULTIPLIERS $\forall$ SINGLE TRAINING SAMPLE
  - THIS LEADS TO = MAXIMIZE STREET WIDTH
- KEEP ONLY SUPPORT VECTORS (TRAIN SAMPLES WITH NON-ZERO α_i) & THEIR α_i
- USE SUP. VEC. & THEIR α_i TO CLASSIFY NEW SAMPLES.

* WE USE HARD MARGIN WHEN TWO CLASSES ARE LINEARLY SEPARABLE
* WE FIND α_i'S SO THAT WIDTH OF // IS MAXIMIZED
* NO TRAINING SAMPLES ARE ALLOWED ON THE WRONG SIDE OF MARGINS

FROM HERE NOW A PROBLEM ARISES:

SOFT MARGINS

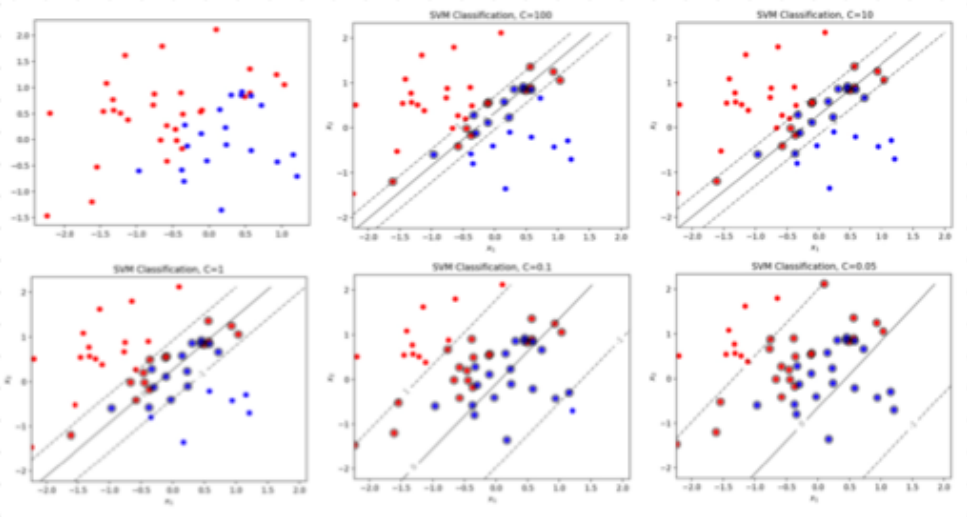
* CONSTRAINT IS RELAXED SO IT IS ALLOWED FOR SOME OF THE TRAINING SAMPLES TO FALL ON WRONG SIDE.
* WE DO ALLOW IT BUT WE PENALIZE IT!!
* OPTIMIZATION = MAXIMIZE STREET WHILE MINIMIZING PENALTY

ONTO THE MATH:

$y(x) = \sum_{i}^{n} \alpha_i y_i (x_i \cdot x) + b \quad 0 \leq \alpha_i \leq C$

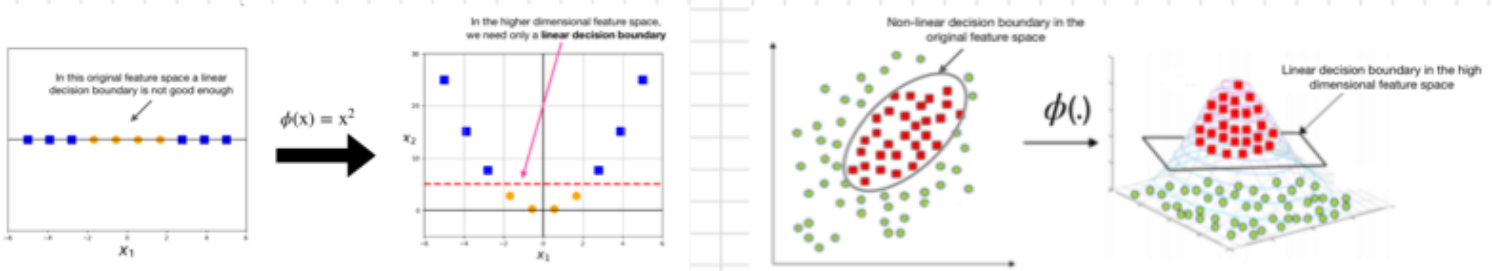
THIS LOOKS THE SAME AS HARD MARGIN EXCEPT THE CONDITION FOR α_i's.

- The hyper parameter $C > 0$ makes a trade-off between the penalty for the samples on the wrong side of the margins & max width
- All samples on the margins or on the wrong side of the margins contribute



SVM with non-linear decision boundaries

1. Sometimes the space might be non-linear:
   - Mapping the features from original dim to higher ones
   - This makes it linear.



A linear decision boundary in the high dim feature space can represent a non-linear decision boundary in original space

1. Some transformation can add many new non-linear features
   - This is due to hopefully allow a good linear decision boundary

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix} \rightarrow \phi(x) = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \\ x_1x_2 \\ \vdots \\ x_{d-1}x_d \\ \vdots \end{bmatrix}$$

Using transformation $\phi(.)$ we map the features from their low dimensional space to a high dimensional space where prob is linear.

model:

$$y(x) = \sum_i^n \alpha_i y_i (\phi(x_i) \cdot \phi(x)) + b$$

1. But a very high dimension means too many computations required for dot products.
   - $\phi(x_i) \cdot \phi(x)$ dot product has a time of order $O(2^d)$

Kernels

1. Makes non-linear decision boundaries easy to create

$$y(x) = \sum_i^n \alpha_i y_i K(x_i, x) + b$$

Kernel Trick

For some transformations $\phi(.)$ there is a kernel function $K(x,z)$ that can take any pair of input vectors & compute the dot product of their transformed version $(\phi(x)\cdot\phi(z))$ directly in the original feature space

We could transform feature vectors to infinite dims & still have no extra computation

We just need to be able to compute $K(x,z)$

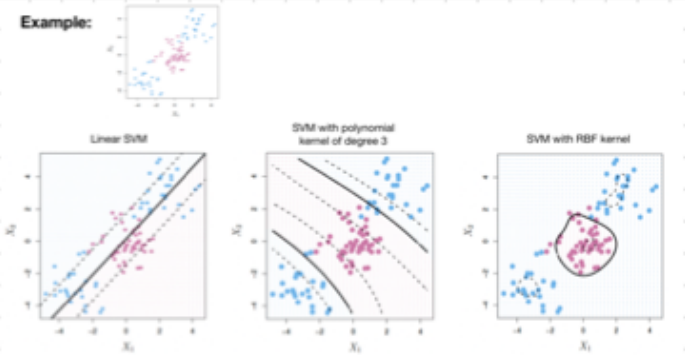
Popular ones:

- Linear Kernel: $K(x,z) = x \cdot z$

Polynomial Kernel: $K(x,z) = (x \cdot z + 1)^b$

:

Example:



Pros vs Cons

Pros

- Effective in high dimensional space
- " when # of features > training examples
- Best algo when classes are separable
- Memory efficient due to support vectors
- outliers have less impact "
- Results can be explained!!

Cons

- Large dataset - takes a lot to train!
- Sensitive to hyper params
- Selecting correct/appropriate kernel