LECTURE 3: CLASSIC ML: LINEAR MODELS

EKATERINA MURAVLEVA

OVERVIEW OF THE COURSE

Lecture 1: General course information, CRISP-DM methodology

Lecture 2: Supervised learning/unsupervised learning. Classification/regression problems. Accuracy metrics (precision, recall, ROC-AUC scores). Concept of loss functions, overfitting / underfitting.

Lecture 3: Classical ML: Linear regression, logistic regression, support vector machine

Lecture 4: Classical ML: Decision trees, random forests, boosting.

Lecture 5: Classical ML: Dimensionality reduction: linear, non-linear methods.

Lecture 6: Classical ML: Clustering methods

Lecture 7: Basic neural networks

Lecture 8: Scalable algorithms

RECAP OF LECTURE 2

- Supervised learning/unsupervised learning.
- Classification/regression problems. Accuracy metrics (precision, recall, ROC-AUC scores)
- Concept of loss functions
- Overfitting / underfitting

PLAN OF LECTURE 3

- Linear regression
- Logistic regression
- Support vector machine
- Kernel trick

‘CLASSICAL MACHINE LEARNING’

In the upcoming lectures, I will cover what is referred to as **classical machine learning,**

The term ‘classic’ is an opposite to the more modern deep-learning based methods.

Most basic model in machine learning is a ‘linear model’. However, those model can be quite complicated, as we will see.

RECAP: SUPERVISED LEARNING

We are given the training set $\{x_i, y_i\}, \quad i = 1, \ldots, M$ and we are trying to fit the model

$$\hat{y}_i = f(x_i, \theta) \approx y_i$$

What is the model we can imagine?

LINEAR MODEL

How to parametrize the function $y = f(x, \theta)$?

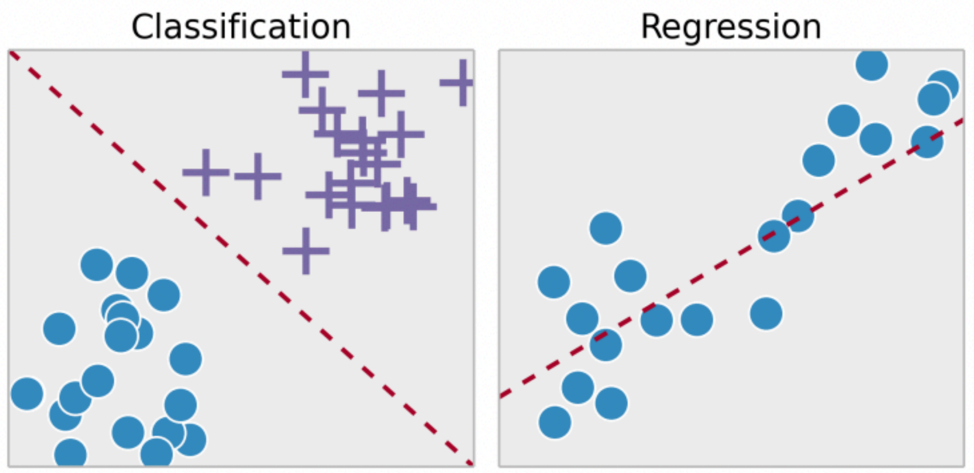
Of course, it depends on the application!

Simplest model is the **linear model**

$$f(x, \theta) = \sum_{j=1}^{n} \theta_j f_j(x) \text{ for regression}$$

$$f(x, \theta) = \text{sign}\left(\sum_{j=1}^{n} \theta_j f_j(x) \right) \text{ for classification}$$

Here $f_j(x)$ are certain **known functions**



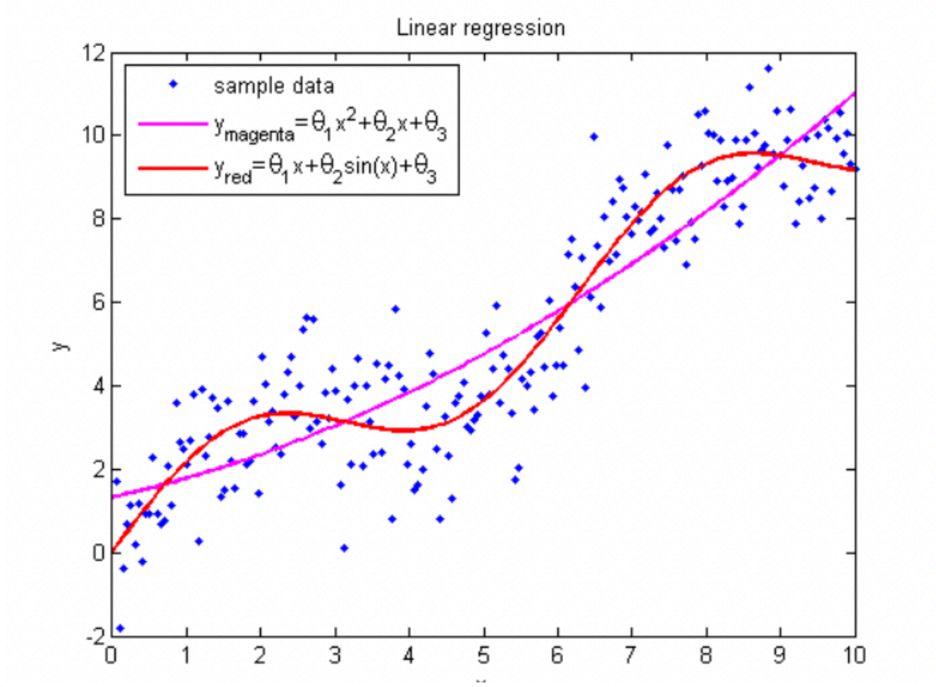
LINEAR REGRESSION MODEL (2)

The choice of functions is very important.

For example, consider the same data, but different features (quadratic model vs model with a sine function)

$X = Y = \mathbb{R}, \quad l = 200, \quad n = 3$, features: $\{x, x^2, 1\}$, or $\{x, \sin x, 1\}$

It is not very easy to understand, which model is better in this case.



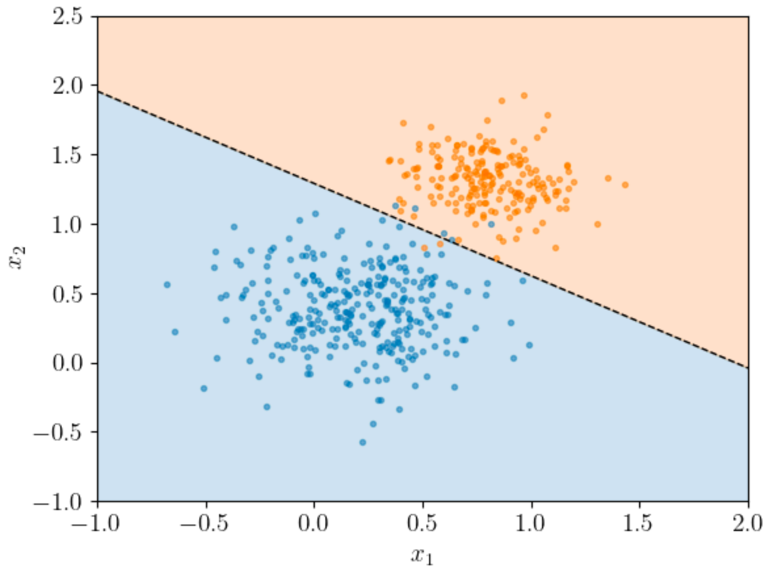
LINEAR MODEL FOR BINARY CLASSIFICATION

In binary classification, we need to separate blue points from orange points using a linear function $f(x, \theta)$

This function has to be > 0 for class 1 (orange) and < 0 for class 0

The set $f(x) = 0$ is called **decision boundary**

In linear classification, f is given a linear function of parameters θ.



SIGMOID FUNCTION (1)

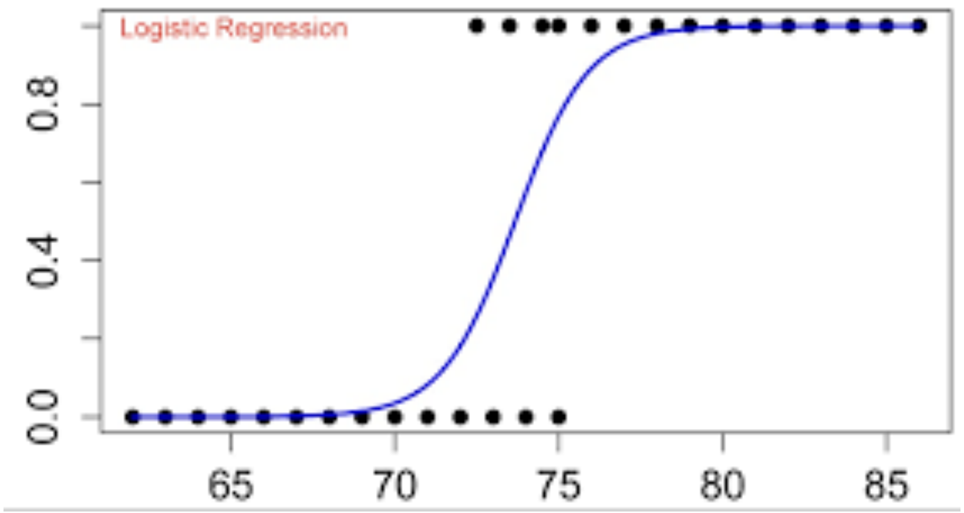
As a loss function, one can use the empirical risk, given as the number of correct predictions.

The problem with such loss is that it is difficult to optimize.

Instead, there are several approaches, the simplest one is to replace a sign function with a smoothed version.

The standard approach uses the sigmoid function

$$\sigma(\hat{y}) = \frac{1}{1 + e^{-\hat{y}}}$$



SIGMOID FUNCTION (2)

The linear model predicts the score function

$\hat{y} = f(x, \theta)$

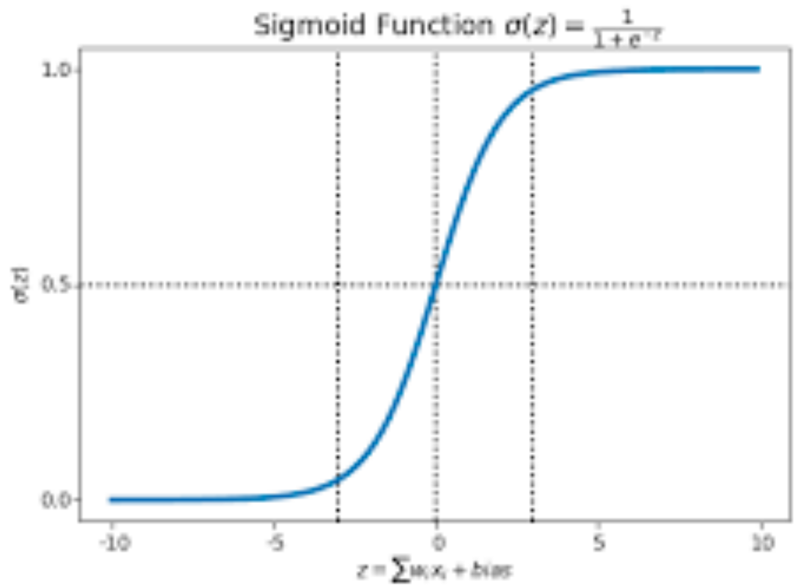
Then, it is put into the sigmoid function

$$\sigma(\hat{y}) = \frac{1}{1 + e^{-\hat{y}}}$$

and we get the value from [0, 1] which is

interpreted as a **probability of predicting class 1.**

I.e if y is negative, $\sigma(y) < \frac{1}{2}$, if y is positive, then $\sigma(y) > \frac{1}{2}$



BINARY CROSS-ENTROPY

The linear model predicts the score function

$\hat{y} = f(x, \theta)$

Then, it is put into the sigmoid function

$$\sigma(\hat{y}) = \frac{1}{1 + e^{-\hat{y}}}$$

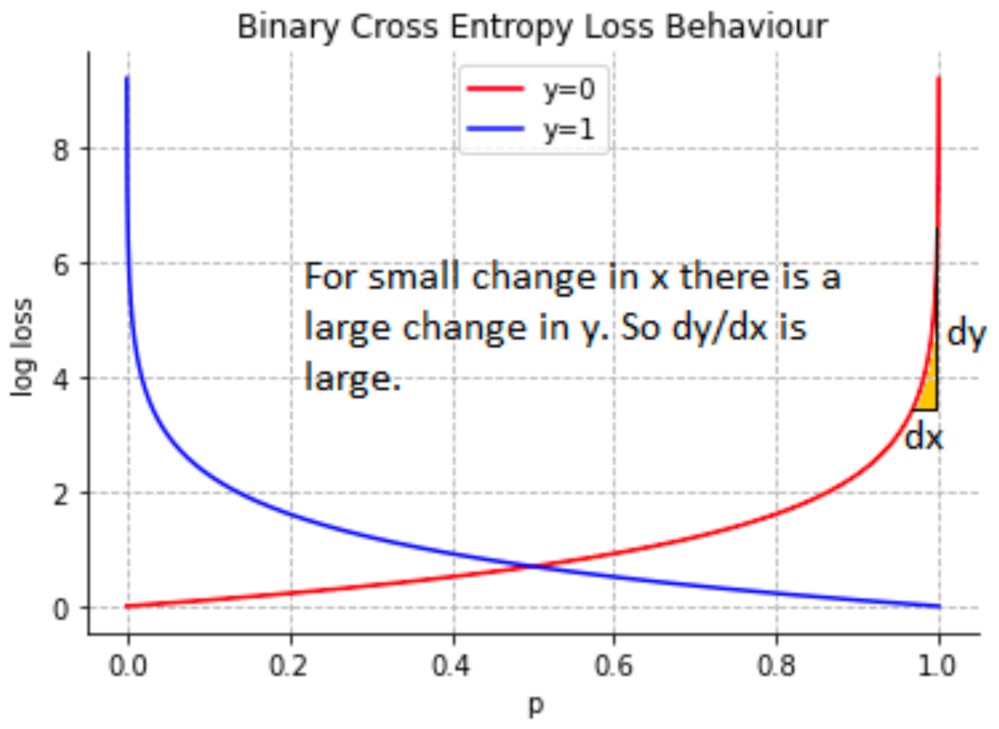
For class 1, we need to maximize $\sigma(\hat{y})$;

For class 0, we need to maximize $1 - \sigma(\hat{y})$

Typically, instead of probability, the logarithm of probability is considered, so we maximize

$$l(y, \hat{y}) = y \log \sigma(\hat{y}) + (1 - y)\log(1 - \sigma(\hat{y}))$$

Here y is the label (0 or 1) and $\sigma(\hat{y})$ is the predicted probability



BINARY CROSS ENTROPY (OR LOGISTIC) LOSS

The final loss is updated by summing all of the individual losses:

$$J(\theta) = -\sum_{i=1}^{N} y_i \log\left(\sigma\left(\hat{y}_i\right)\right) + \left(1 - y_i\right) \log\left(1 - \sigma\left(\hat{y}_i\right)\right) \rightarrow \min$$

Note the minus on the left!

Now we are left with optimization problem with respect to the parameters θ of the linear model for $\hat{y}$

$$\hat{y} = \sum_{j=1}^{n} f_j(x)\theta_j$$

There is no explicit formula, but it can be shown, that this is a

convex optimization with respect to θ

GRADIENT DESCENT

One can use **gradient descent** or **stochastic gradient descent** to update parameters.

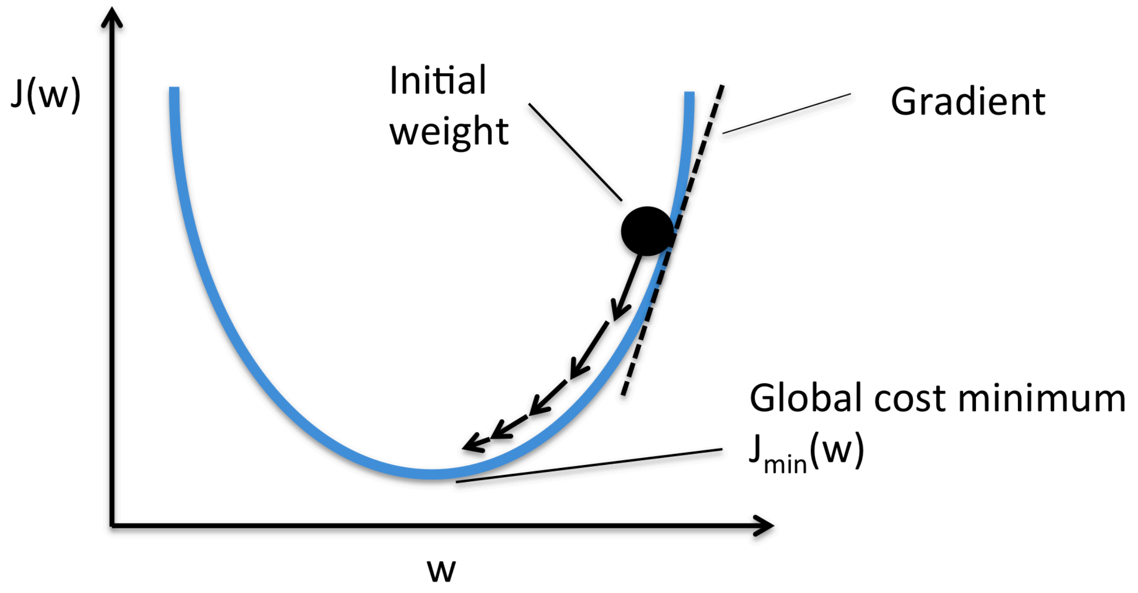
In its simplest form, the update is

$$\theta := \theta - \alpha \frac{\partial J}{\partial \theta}$$

One can evaluate the gradients very efficiently.

In the stochastic gradient descent, we only use a few examples at each update.

For convex optimization, most of the results on convergence and optimal methods are well known.



LINEAR MODEL FOR MULTI-CLASS CLASSIFICATION

Consider multi class classification with K classes. Then, instead of predicting the probability of class 1, we predict the probabilities of K classes simulatenously.

The model predicts K scores (vector $\hat{y}$). In order for them to be interpreted as probabilities, they need to be:

a) non-negative

b) sum up to one

This can be achieved by taking exponent and dividing by sum — **softmax operation**

$$\hat{y} = f(x, \theta), \quad p = \text{softmax}(\hat{y})$$

$$p_i = \frac{e^{\hat{y}_i}}{\sum_{j=1}^{K} e^{\hat{y}_j}}$$

CROSS-ENTROPY LOSS

The generalization of binary cross-entropy to cross entropy is done straightforwardly:

We maximize the probability of the true class!

$$\hat{y} = f(x, \theta), \quad p = \mathrm{softmax}(\hat{y})$$

$$p_i = \frac{e^{\hat{y}_i}}{\sum_{j=1}^{K} e^{\hat{y}_j}}$$

Let y_i be the one-hot encoding of the label (vector of length K).

Then, $l(y, \hat{y}) = \sum_i y_i \log p_i \to \max$

Question for understanding: what happens if K = 2?

MULTINOMIAL LOGISTIC REGRESSION: SUMMARY

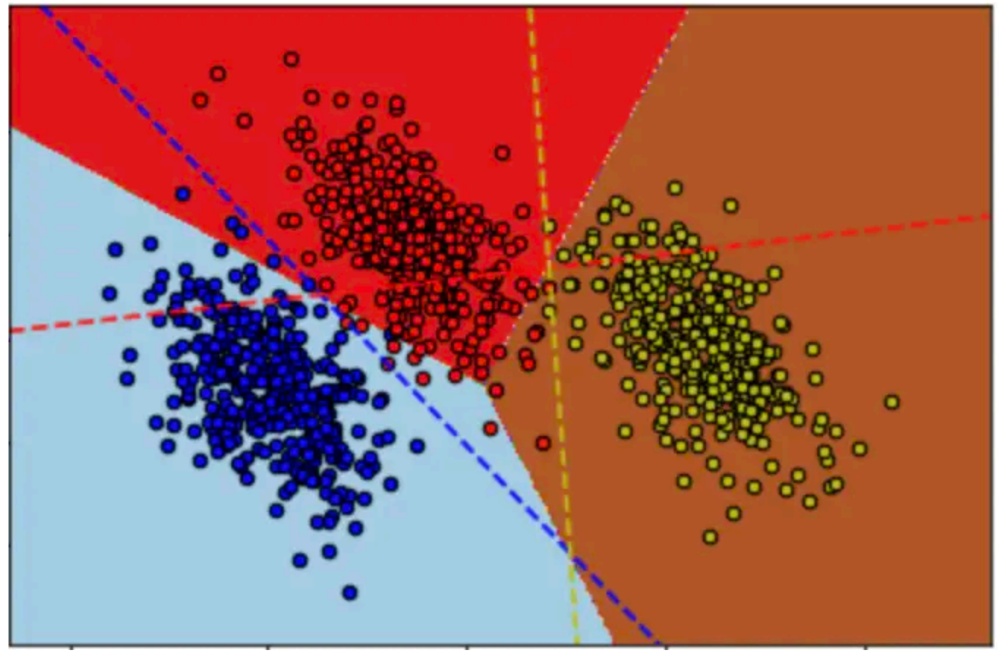
We can use linear mapping + one-hot encoding + cross-entropy loss.

For a linear model, the problem is still convex

Can be solved by optimization techniques for convex problems.

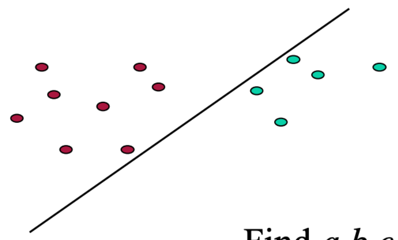
In practice, often run stochastic gradient descent

Instead of one decision boundary, we have many!



LINEAR MODEL FOR CLASSIFICATION: SUPPORT VECTOR MACHINE

For linear support vector machine we want to find a **separating hyperplane** for two classes.



Find a,b,c, such that

$ax + by \geq c$ for red points

$ax + by \leq$ (or $<$) c for green points.

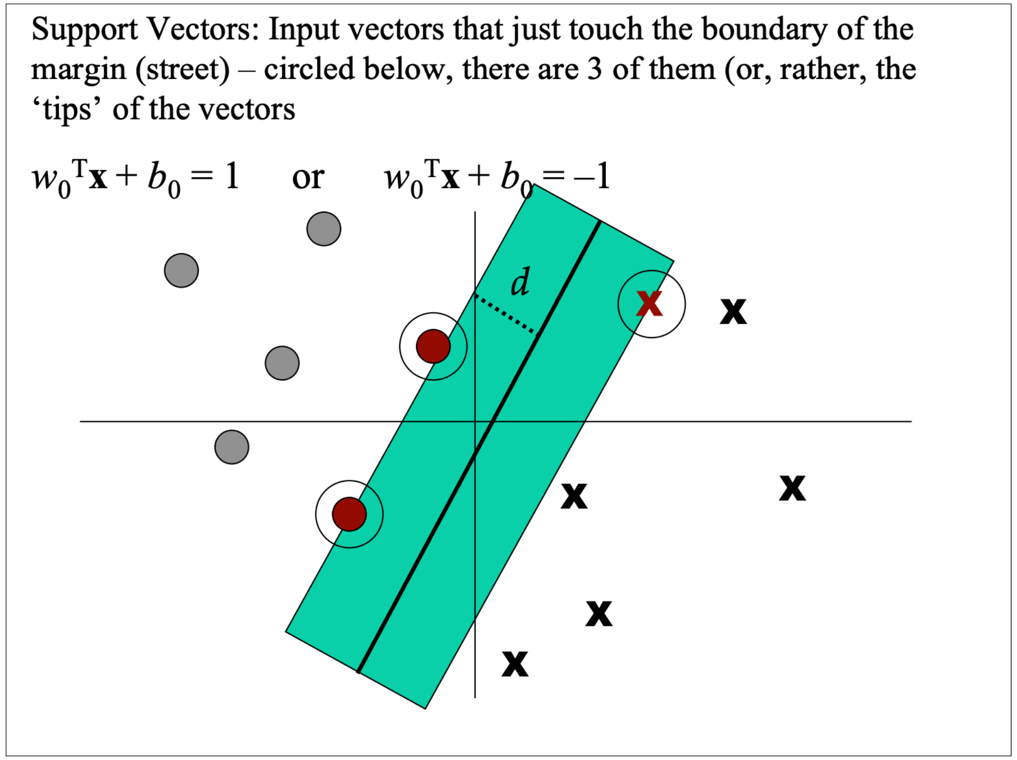
LINEAR MODEL FOR CLASSIFICATION: SUPPORT VECTOR MACHINE

For linear support vector machine we want to find a **separating hyperplane** for two classes.

Consider 2D case.

A lot of possible solutions for a, b, c

Support vector is an element of the training set that would change position of the hyperplane if removed.


Support Vectors: Input vectors that just touch the boundary of the margin (street) – circled below, there are 3 of them (or, rather, the 'tips' of the vectors

$w_0^T\mathbf{x} + b_0 = 1$ or $w_0^T\mathbf{x} + b_0 = -1$

LINEAR MODEL FOR CLASSIFICATION: SUPPORT VECTOR MACHINE

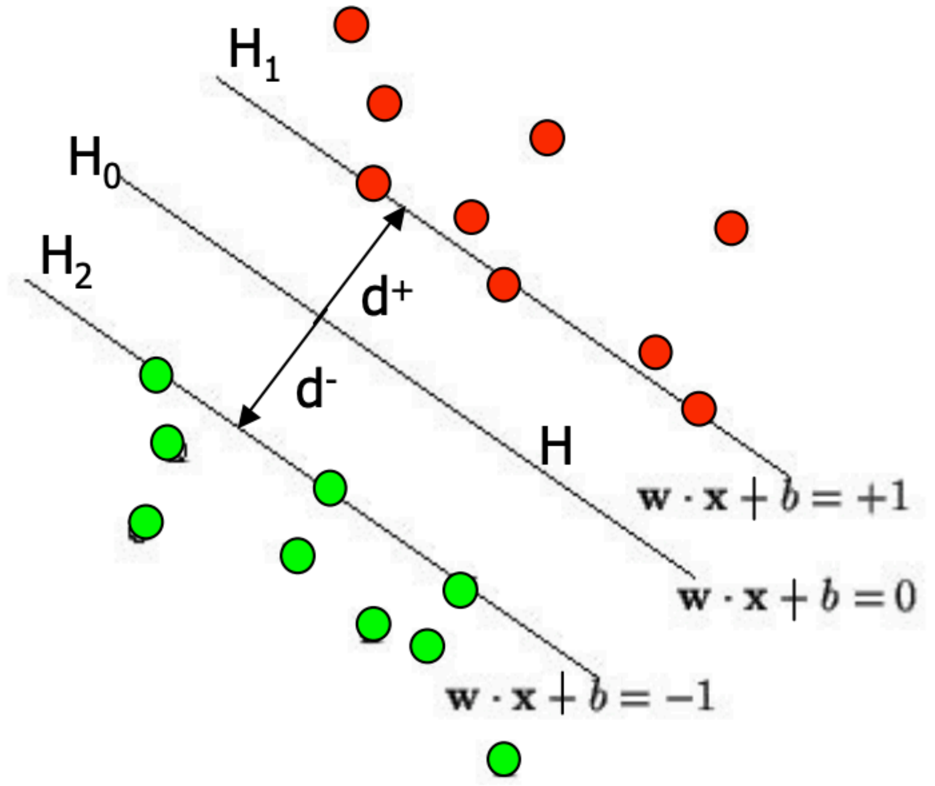
Define hyperplanes such that

$(w, x_i) + b \geq +1$ when $y_i = +1$

$(w, x_i) + b \leq -1$ when $y_i = -1$

The plane H_0 is the median in-between, where

$(w, x) + b = 0$


H_1
H_0
H_2
d^+
d^-
H
$\mathbf{w} \cdot \mathbf{x} + b = +1$
$\mathbf{w} \cdot \mathbf{x} + b = 0$
$\mathbf{w} \cdot \mathbf{x} + b = -1$

MAXIMIZING THE MARGIN

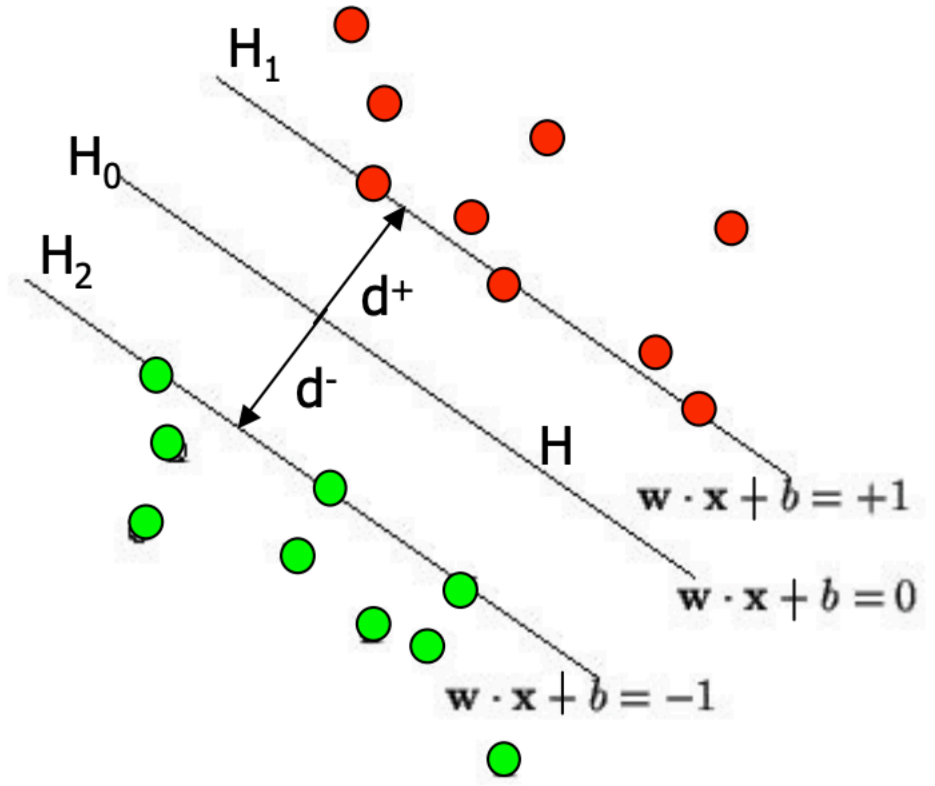
We want a classifier (=linear separator) with as big margin, as possible.

The distance between a point (x_0, y_0) and a line $Ax + By + c$ is (elementary geometry)

$$\frac{|Ax_0 + By_0 + c|}{\sqrt{A^2 + B^2}}$$

So the distance between H_0 and H_1 is $\frac{1}{\|w\|}$ and the total distance between H_1 and H_2 is $\frac{2}{\|w\|}$

To maximize the margin, we need to minimize $\|w\|$


H_1
H_0
H_2
d^+
d^-
H
$\mathbf{w} \cdot \mathbf{x} + b = +1$
$\mathbf{w} \cdot \mathbf{x} + b = 0$
$\mathbf{w} \cdot \mathbf{x} + b = -1$

SVM LOSS FUNCTION

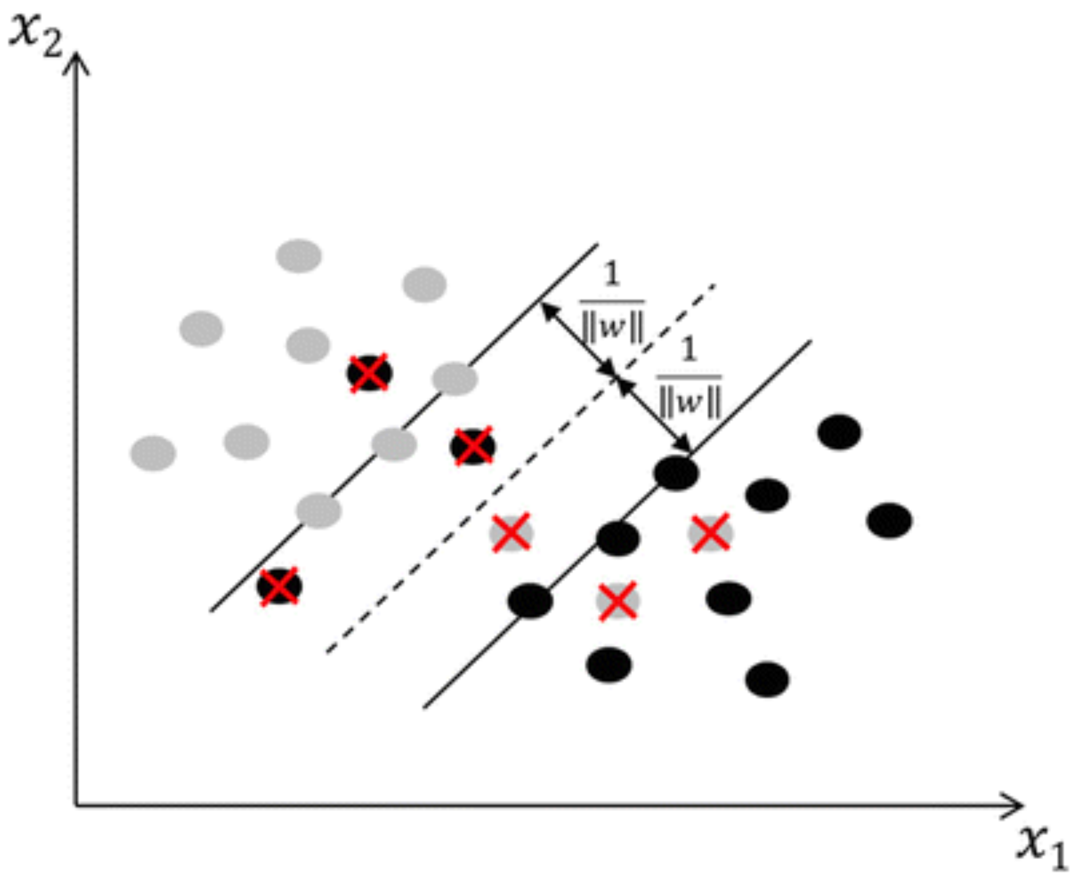
The final loss function for SVM has the form

$$1/2||w||^2 + C\sum_i (\max(0, 1 - y_i(w^T x_i + b)) \to \min$$

Where C is the regularization parameter which controls the misclassification (indeed, if the two sets are not separable, we need to regularize

The problem is convex, but non-smooth.

Also, we only have scalar products of (w, x_i) in this formulation!



GOING FROM LINEAR MODEL TO NON-LINEAR MODEL

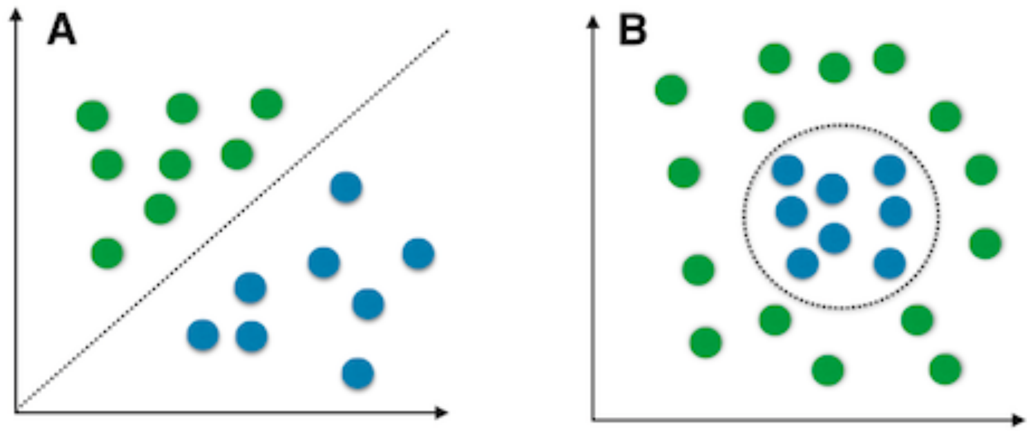
What happens, if the data is not linearly separable?

Linear decision boundary can not split the data into two classes.

We need to use non-linear decision boundary.

Question: How we can do that, while still leaving model linear?

Linear vs. nonlinear problems



HIGH-DIMENSIONAL FEATURE MAP

Lets get back to the start of the lecture, where we considered the following function:

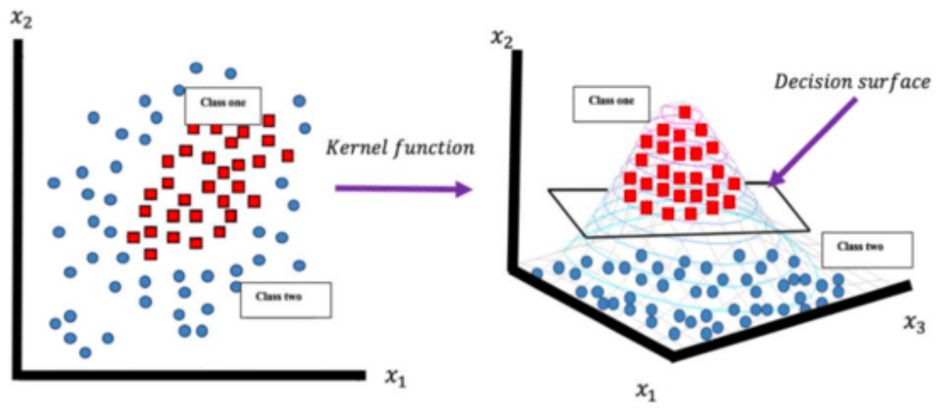
$$f(x,\theta) = \sum_{j=1}^{n} \theta_j f_j(x)$$

We can say that instead of features x we use features $f_1(x), f_2(x)\ldots, f_n(x)$

With such features, the dataset can become linearly separable!

However, if n is large, linear classification becomes more and more complicated.

Here where **kernel trick** emerges.



KERNEL TRICK

Instead of working with **explicit feature map**

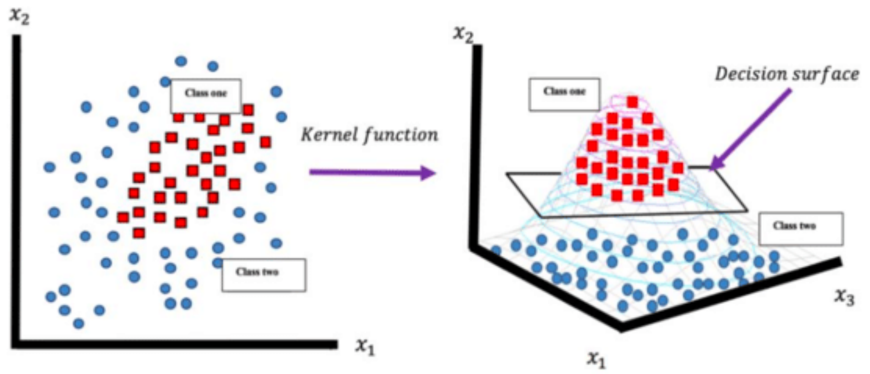
$[g(x_i)]_j = f_j(x_i)$ (we will need to store n coefficients then)

We can work with scalar products

$$(g(x_i), g(x_k)) = K(x_i, x_k)$$

It can be shown, that all of the loss functions can be written in terms of the kernel function,

we don't need the explicit feature map!



HOW WE CAN TRANSFORM THE PROBLEM

This is called the **primal problem** (only for the linearly separable data!!!!)

$$\underset{\boldsymbol{w}, w_0}{\text{minimize}} \frac{1}{2} \|\boldsymbol{w}\|_2^2$$

subject to $y_j \left(\boldsymbol{w}^T \boldsymbol{x}_j + w_0 \right) \geq 1, \quad j = 1, \ldots, N.$

It can be shown, that it can be reformulated as a **dual problem** (highly non-trivial derivation!, see https://engineering.purdue.edu/ChanGroup/ECE595/files/Lecture20_SVM2.pdf)

$$\underset{\lambda \geq 0}{\text{maximize}} -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j x_i^T x_j + \sum_{j=1}^{N} \lambda_j$$

subject to $\sum^{N} \lambda_j y_j = 0.$

$\boldsymbol{w}^* = \sum_{j \in \mathcal{V}} \lambda_j y_j \boldsymbol{x}_j$ where $\mathcal{V}$ is the set of non-zeros in λ

FINALLY! KERNEL TRICK!

$$\underset{\lambda\geq 0}{\text{maximize}} - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\lambda_i\lambda_j y_i y_j x_i^T x_j + \sum_{j=1}^{N}\lambda_j$$

subject to $\sum^{N} \lambda_j y_j = 0.$

We can replace $x_i^\top x_j$ with $K(x_i, x_j)$ and get the non-linear SVM.

Question: How to get the decision boundary from λ?

STANDARD KERNELS

Kernel function $K(x, y)$ has to be **positive definite** (generalization of positive definite matrices)

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

holds for any $x_1, \ldots, x_n \in \mathcal{X}$, given $n \in \mathbb{N}, c_1, \ldots, c_n \in \mathbb{R}$.

Examples:

1. Linear kernel: $K(x_i, x_j) = x_i^T x_j$

2. Polynomial kernel: $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d$

3. Gaussian (RBF) kernel: $K(x_i, x_j) = \exp(-\gamma || x_i - x_j ||^2)$

KERNEL METHODS: PROS AND CONS

1. Can handle complex non-linear decision boundaries
2. Need to work with a kernel matrix of size $N \times N$, where N is the size of the dataset.
3. Complexity typically grows as $\mathcal{O}(N^3)$. Good for small datasets, can not scale easily to million-sized dataset.
4. Some of those problems can be solved by **inverting the kernel trick** , but it is an advanced topic (see Random Fourier Features, if interested).

KERNEL TRICK FOR REGRESSION

One can easily apply kernel trick for regression.

In fact, it is equivalent to the linear model of the form

$f(x, \theta) = \sum_{i=1}^{N} \theta_i K(x, x_i)$, i.e. the linear model with dimension equal to the number of samples in the dataset.

Instead of solving least squares problems, one has to carefully **regularize** the solution

RECAP OF LECTURE 3

- Linear regression
- Logistic regression
- Support vector machine
- Kernel trick

NEXT LECTURE

Classical ML:

Decision trees,

random forests,

boosting.