Course:CM-072

Class 2

List of topics

- Linear models
  - Linear Regression
  - Regularization L1 and L2
  - Ridge Regression
  - LASSO Regression

- Classification with Linear Models
  - Logistic Regression
  - Naive Bayes classifier
  - SVC
- Linear models for multiclass classification
  - One vs. All classification

Linear Models

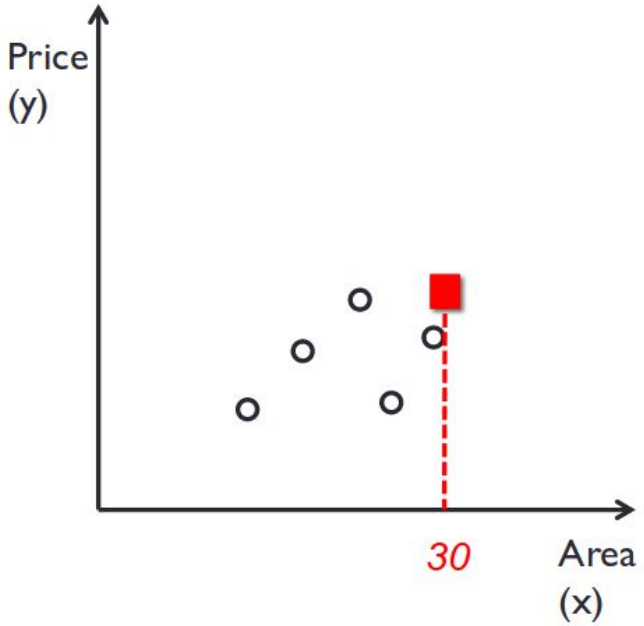
Linear Model is a family of model-based learning approaches that assume the output y can be expressed as a linear algebraic relation with the input attributes $\mathbf{x_1}$, $\mathbf{x_2}$ **...**

The input attributes $\mathbf{x_1}$, $\mathbf{x_2}$ **...** is expected to be numeric and the output is expected to be numeric as well.

The most common model which is linear is **Linear Regression**.

Univariate Linear Regression

What is the estimated price of a new house of area 30 m2?



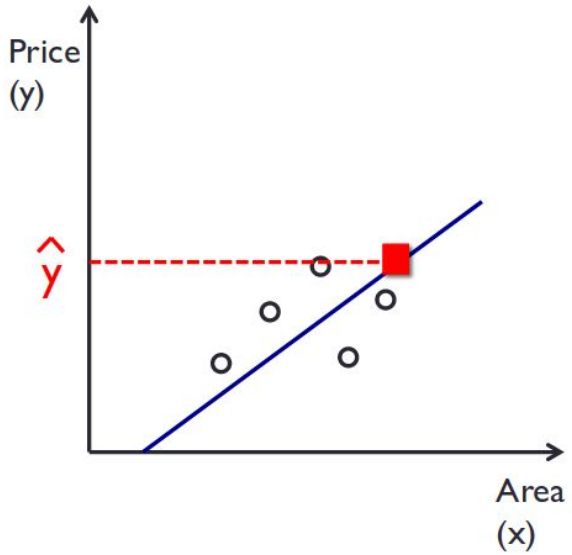
Univariate Linear Regression

Find the line (or linear function) to fit the input points:

y = ax+b

Univariate Linear Regression

What is the estimated price of a new house of size 30?



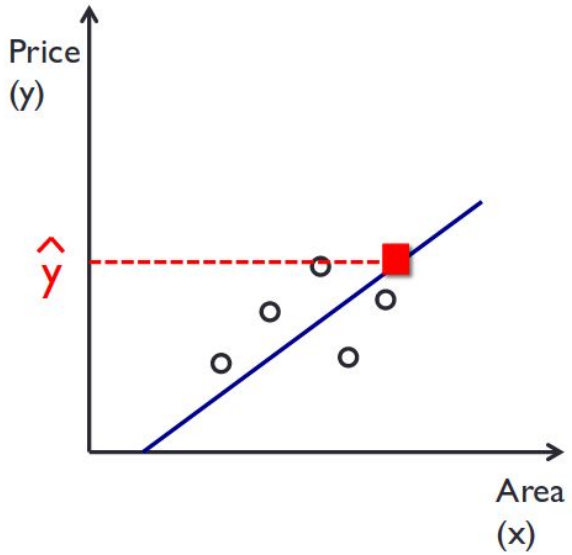
Univariate Linear Regression

We must calculate:

$$y = ax + b$$

coefficient
(the slope of the line)

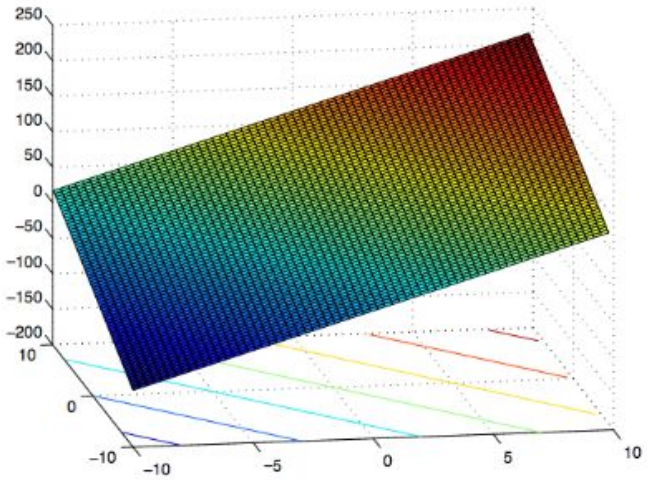
intercept
(the offset of the line, ie the point where it hits the y axis)



Multivariate Linear Regression

In general:

$$h(\boldsymbol{x}) = w_1 x_1 + w_2 x_2 + \ldots + w_2 x_2 + b$$



Example of a plane

h(x) is called a line for one feature, a plane for two features or a hyperplane for more features.

Multivariate Linear Regression

$$h(\mathbf{x}) = w_1x_1+w_2x_2+\ldots+w_nx_n+ b$$

is called the hypothesis and it can also be written as: $h(\mathbf{x}) = \sum_{i=0}^{n} \mathbf{w}[i]\mathbf{x}[i] = \mathbf{w}^T\mathbf{x}$

To obtain this, we have defined one extra feature: $x_0 = 1$ in order to represent **b** as w_0 :

$$\mathbf{x} = (x_0,\ldots, x_n) \in \mathcal{R}^n$$

$$\mathbf{w} = (w_0,\ldots, w_n) \in \mathcal{R}^n$$ weight vector

Multivariate Linear Regression

$$h(\mathbf{x}) = w_1x_1 + w_2x_2 + \ldots + w_nx_n + b$$

is called the hypothesis and it can also be written as:

$$h(\mathbf{x}) = \sum_{i=0}^{n} \mathbf{w}[i]\mathbf{x}[i] = \mathbf{w}^T\mathbf{x}$$

1, (n+1) matrix

(n+1),1 matrix

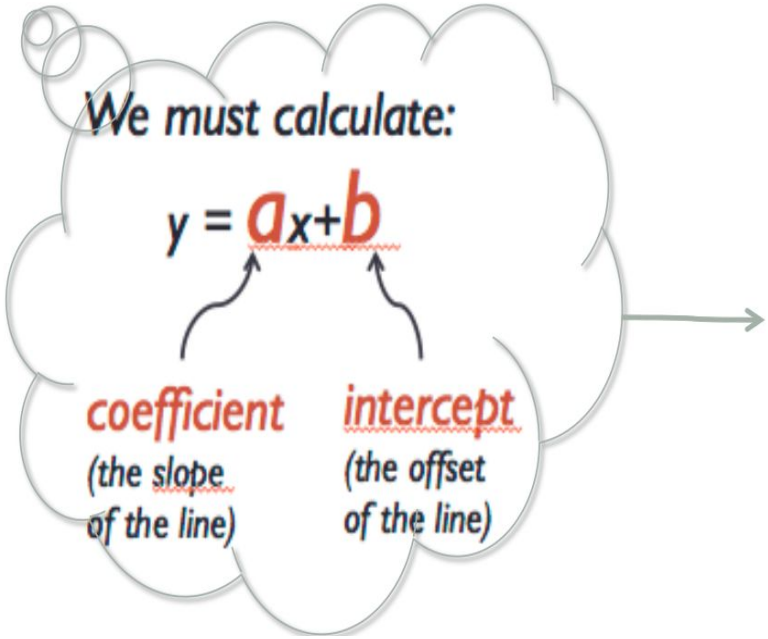
Inner product of the transpose of **w** and **x**

To obtain this, we have defined one extra feature: $x_0 = 1$ thus:

$$\mathbf{x} = (x_0, \ldots, x_n) \in \mathcal{R}^n$$

$$\mathbf{w} = (w_0, \ldots, w_n) \in \mathcal{R}^n$$ weight vector

Multivariate Linear Regression



$$h(\mathbf{x}) = \sum_{i=0}^{n} \mathbf{w}[i]\mathbf{x}[i] = \mathbf{w}^T\mathbf{x}$$

We must calculate **w**

- w[0] is the intercept and
- w[1],…,w[n] are the coefficients

Goal?

Minimize a cost function!

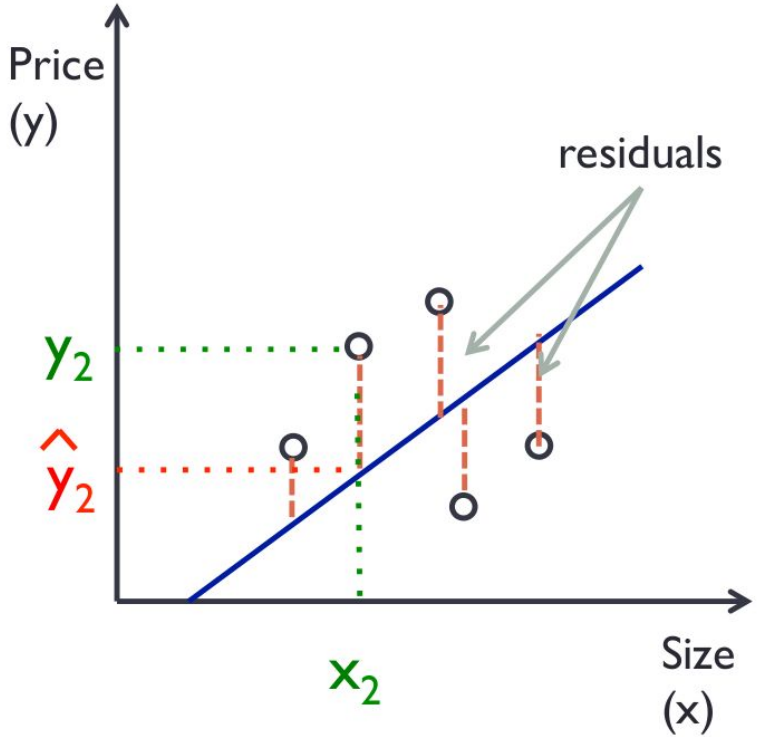
Cost function

The cost function calculates the error.

Minimize the error can approximately be seen as:

Minimize the sum of the squares of the distance of the real values from the estimated values on the training data.

$y - \hat{y}$ is also called residual.



Reminder: $\hat{y}$ denotes an estimated (predicted) value.

Cost function-Ordinary Least Squares

Residual Sum of Squares (RSS)

$$J(\mathbf{w}) = \sum_{i=1}^{N} (h(\mathbf{x}^i) - y^i)^2$$

Goal: *Find* $\mathbf{w}=(w_0,\ldots,w_n)$ *s.t.* $J(\mathbf{w})$ *is minimized.*

- How? Gradient Descent.

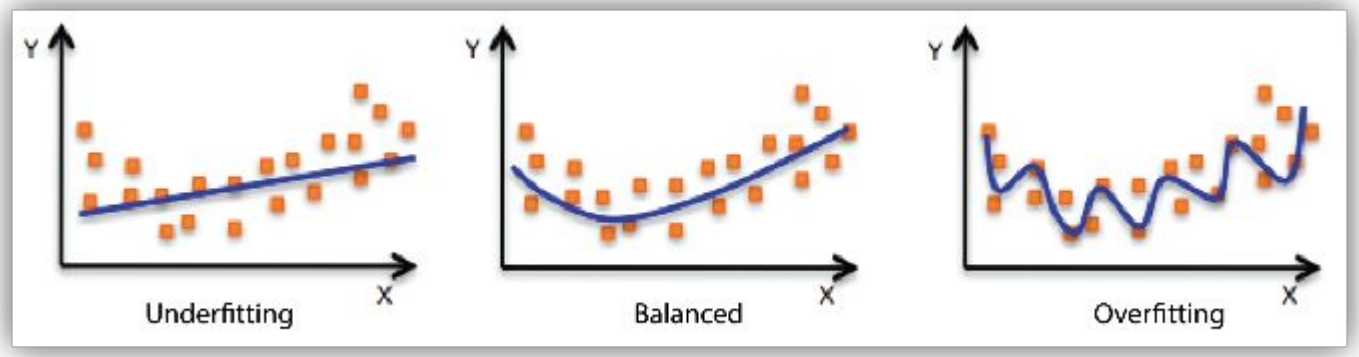
- Intuition: We want to obtain predicted values as close as possible to the original values (for the training data).
- Simple linear regression uses this method, called **Ordinary Least Squares.**

Ordinary Least Squares

- **Pros:**
  - Has no parameters to tune.
  - Performs well in bigger datasets.
  - Simple algorithm to understand.
- **Cons:**
  - Performs badly for small size of input data and small number of features.
  - Can be slow in big datasets.
  - High model complexity, as all the features are taken into account.
  - Risk of over- or underfitting.

Ordinary Least Squares

- Underfitting may occur when we have few features and few training data.
- Overfitting may occur when we have many features (eg in the last diagram the hyperplane is projected on the x feature).
- To avoid overfitting: Intervene by **regularization!.**



Notions of Regularization

There are few ways you can avoid overfitting of a model on a training data like cross-validation sampling, reducing number of features, pruning, **regularization** etc.

Regularization basically adds the penalty as model complexity increases. Regularization parameter (alpha) penalizes all the parameters except intersect so that model generalizes the data and won't overfit.

A regression model that uses L1 regularization technique is called ***Lasso Regression*** and model which uses L2 is called ***Ridge Regression***.

The key difference between these two is the penalty term.

Ridge Regression

Characteristics:

- Tries to **shrink** the estimated weights to zero.
- **L2 penalty:** Penalizes the L2 norm (euclidean length) of the coefficient vector.
- A tuning parameter **α** controls the strength of the penalty.

Now, the cost function becomes:

$$J^r(\mathbf{w}) = \underbrace{\sum_{i=1}^{N}(h(\mathbf{x}^i) - y^i)^2}_{\text{error}} + \underbrace{\alpha \sum_{j=1}^{n} \mathbf{w}_j^2}_{\text{penalty}}$$

Goal: *Find* $\mathbf{w}=(w_0,\ldots,w_n)$ *s.t.* $J^r(\mathbf{w})$ *is minimized.*

Ridge Regression

Characteristics of parameters:

$$J^r(\mathbf{w}) = \underbrace{\sum_{i=1}^{N}(h(\mathbf{x}^i) - y^i)^2}_{\text{error}} + \underbrace{\alpha \sum_{j=1}^{n} \mathbf{w}_j^2}_{\text{penalty}}$$

The tuning parameter $\alpha \geq 0$ controls the strength of the penalty.

- $\alpha = 0$: we get **linear regression.**
- $\alpha \rightarrow \infty$: we get $w \rightarrow 0$
- For other values of α: we try to fit a linear model h(x) and in the same time shrink the weights.
  - Weights here are only the **coefficients** of the features, not the intercept-note how we start from j=1 and not j=0.

Lasso Regression

Characteristics:

- Ridge regression will never set the coefficients to zero only when **α** is **∞** will all coefficients be 0.
- What if some features are not as important as others? **Feature Selection**.
- Regularized model that penalizes the L1 norm of the coefficient vector.
- A tuning parameter **α** controls the strength of the penalty.
- Now, the cost function is:

$$J^{l}(\mathbf{w}) = \sum_{i=1}^{N} (h(\mathbf{x}^{i}) - y^{i})^{2} + \alpha \sum_{j=1}^{n} | \mathbf{w}_{j} |$$

Lasso Regression

Characteristics of parameters:

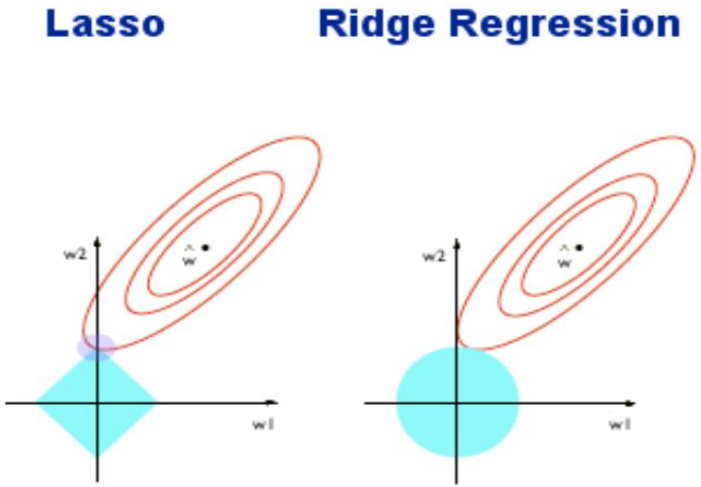
$$J^{l}(\mathbf{w}) = \sum_{i=1}^{N} (h(\mathbf{x}^{i}) - y^{i})^{2} + \alpha \sum_{j=1}^{n} |\mathbf{w}_{j}|$$

As for Ridge regression, for the tuning parameter $\alpha \geq 0$ controls the strength of the penalty.

- $\alpha = 0$: we get **linear regression.**
- $\alpha \rightarrow \infty$: we get $w \rightarrow 0$
- For other values of α: we try to fit a linear model h(x) and in the same time shrink the weights.
- Which can now be exactly 0 for some features, because of L1.
- For bigger values of α, more weights become 0 and the rest are getting smaller.

Lasso Regression

Why does Lasso give some 0 weights?



For Lasso the green region represents the constraint area imposed ny L1: $|w_1+w_2| \leq t$

For Ridge the constraint region is: $w_1^2 + w_2^2 \leq t$

The RSS has elliptical contours, centered at the OLS estimate $\hat{w}$.

We can see that the lasso diamond has corners, enabling some coefficients (here w1) to become zero when the elliptical contour of RSS meet the green contour.

Summary

- **Ridge and Lasso**
  - Regularized models (using L2 and L1 penalties respectively).
  - Better prediction error than linear regression for small datasets.
  - Needs attribute normalization.
- **Ridge**
  - Works best when a number of weights are already small or 0.
  - Is good for prediction– not for model interpretation (no feature selection is possible).
- **Lasso**
  - Makes feature selection possible.
  - Good for model prediction.
- Combinations of L1 and L2 also exist (elastic net).

Classification with Linear Models

From Bishop Books Pattern Recognition and Machine Learning:

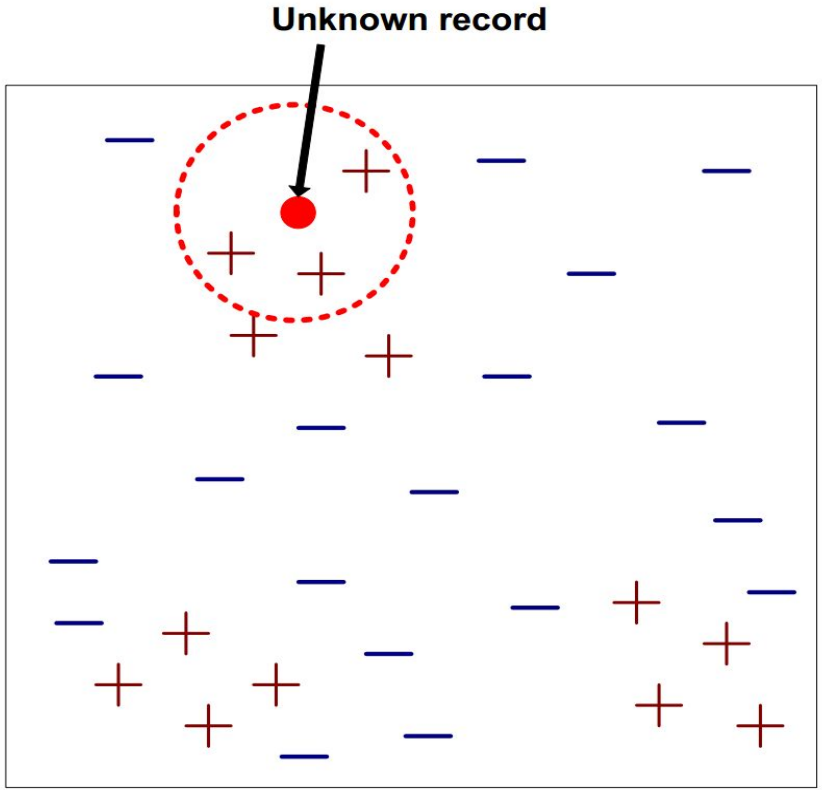
The goal in classification is to take an input vector **x** and to assign it to one of K discrete classes C_K where k = 1, . . . , K. In the most common scenario, the classes are taken to be disjoint, so that each input is assigned to one and only one class. The input space is thereby divided into **decision regions** whose boundaries are called **decision boundaries** or **decision surfaces**.

We consider linear models for classification, by which we mean that the **decision surfaces are linear functions** of the input vector **x** and hence are defined by (D − 1)-dimensional hyperplanes within the D-dimensional input space. Data sets whose classes can be separated exactly by **linear decision surfaces** are said to be **linearly separable.**

k-Nearest Neighbor (k-NN)

- **The simplest classification algorithm**
- **Some characteristics:**
  - Performs well in low dimensional spaces. Curse of dimensionality!.
  - Scalability issues.
  - Needs attribute reformatting (to numeric) and scaling.
- **Lazy learner (vs Eager learner)**
  - Stores the input (training) data.
  - Does not construct a learning model.
  - ‘Learns’ the class or the target value of a new data point by comparing it to the stored data.

k-Nearest Neighbor (k-NN) Classifier



What is the class of the bullet?

A plus or a minus?
(binary classification)

The neighbors will tell us!.

k-Nearest Neighbor (k-NN) Classifier

Input:

- $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$: training (n-dimensional) vectors **X** and corresponding labels Y.
- k: number of neighbors.
- d: distance metric.

Process: Given a new point **x:**

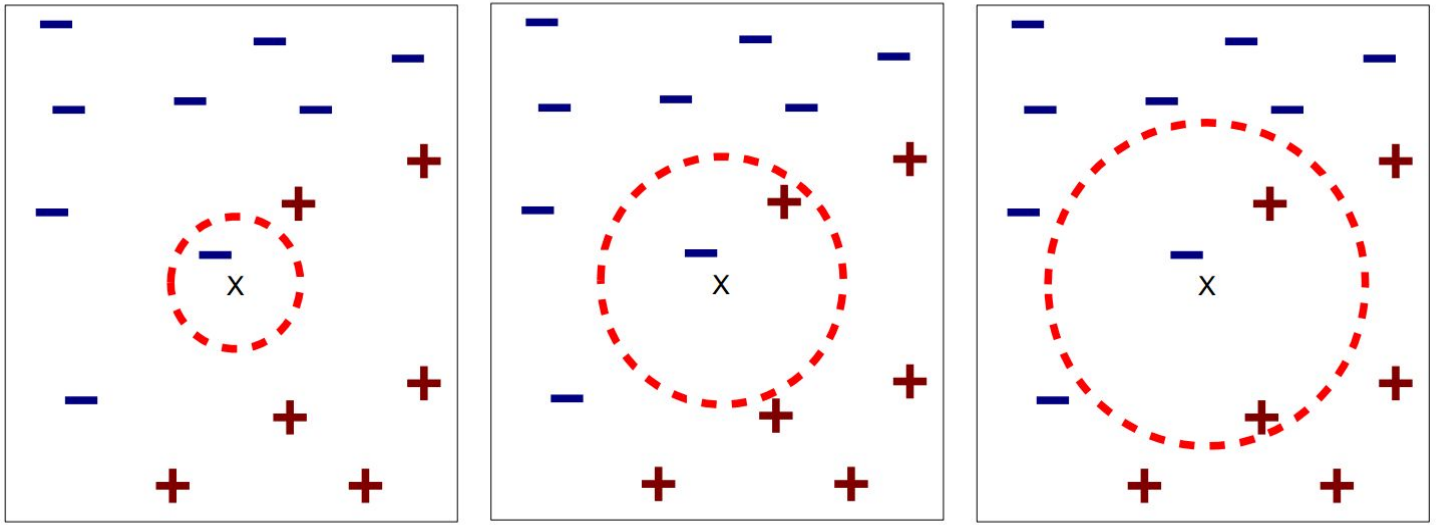
- find the k nearest points $\mathbf{x}_i$ in **X** based on d.
- find the most popular class y_i among $\mathbf{x}_i$ (**majority vote**).
- assign y_i to **x**.

Output: the estimated class for **x**, $y = y_i$.

k-Nearest Neighbor (k-NN) Classifier

The NN rule:

Find the point y in **X** which is nearest to **x**. Assign the label of y to **x**.

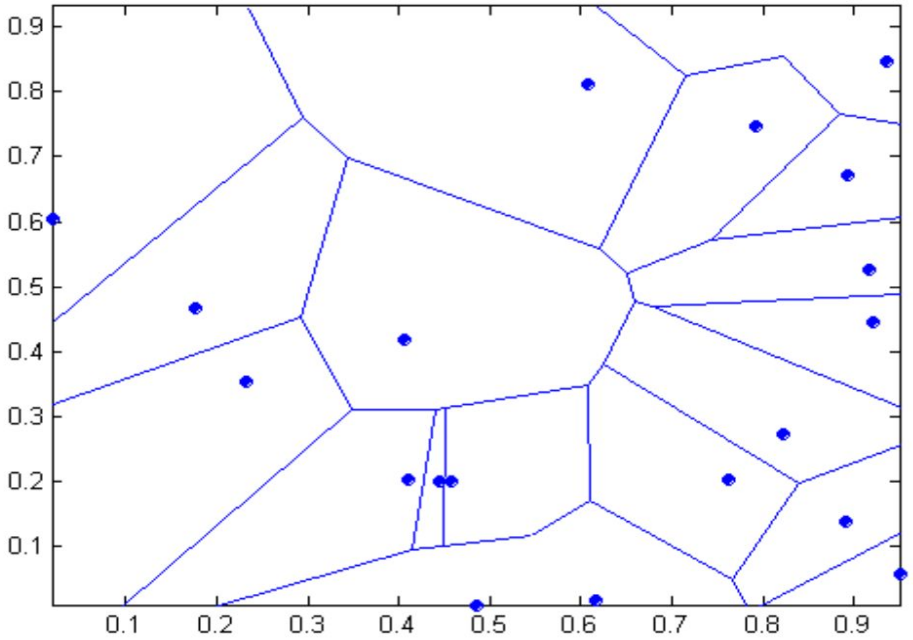


(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

1-Nearest Neighbor

This rule allows us to partition the feature space into cells consisting of all points closer to a given training point **x** . All points in such cells are labeled by the class of the training point. This partitioning is called of **Voronoi**.

2D Voronoi Partition



Distance metrics-numeric

- Let **x** = (x^1,..., x^n) and **x'** = (x'^1,..., x'^n) be two vectors in the n-dimensional datasets.
- MInkowski distance of order p.

$$d = (\sum_{i=1}^{n} | x^i - x'^i |^p)^{1/p}$$

1. For p=2: ***Euclidean distance***.
2. For p=1: Manhattan distance.

Distance metrics-categorical

Two choices:

- Reformat to numeric attributes
- Use the Hamming Distance on categorical attributes. For two points x and x' and an attribute at index i, the distance is defined as

$$d_i(x, x') = d(x^i, x'^i) = \begin{cases} 0 & \text{if } x^i = x'^i \\ 1 & \text{otherwise} \end{cases}$$

Then, the total hamming distance between x and x' is:

$$d(x, x') = \sum_{i=1}^{n} d_i(x, x')$$

k-NN Classification decision

Let x_i denota a k-nearest neighbour of **x**. Then the class y of **x** is:

- **Majority vote:**

$$y = \underset{y'}{\operatorname{argmax}}(|\{x_i : class(x_i) = y'\}|)$$

- **Weighted by distance:**

$$y = \underset{y'}{\operatorname{argmax}}\Big(\sum_{\substack{d_i : d_i = d(x_i, x) \wedge \\ class(x_i) = y'}} \frac{1}{d_i}\Big)$$

k-Nearest Neighbor (k-NN) Regressor

- Same philosophy as with the classifier: Find k nearest neighbors.
- The predicted value is the **mean** of the target values of the k neighbors.

(3-NN) Regressor House Price Index (HPI)

- **Example**

Loan	HPI
40000	135
60000	256
80000	231
20000	267
120000	139
18000	150
95000	127
62000	216
100000	139
220000	250
150000	264
142000	**?**

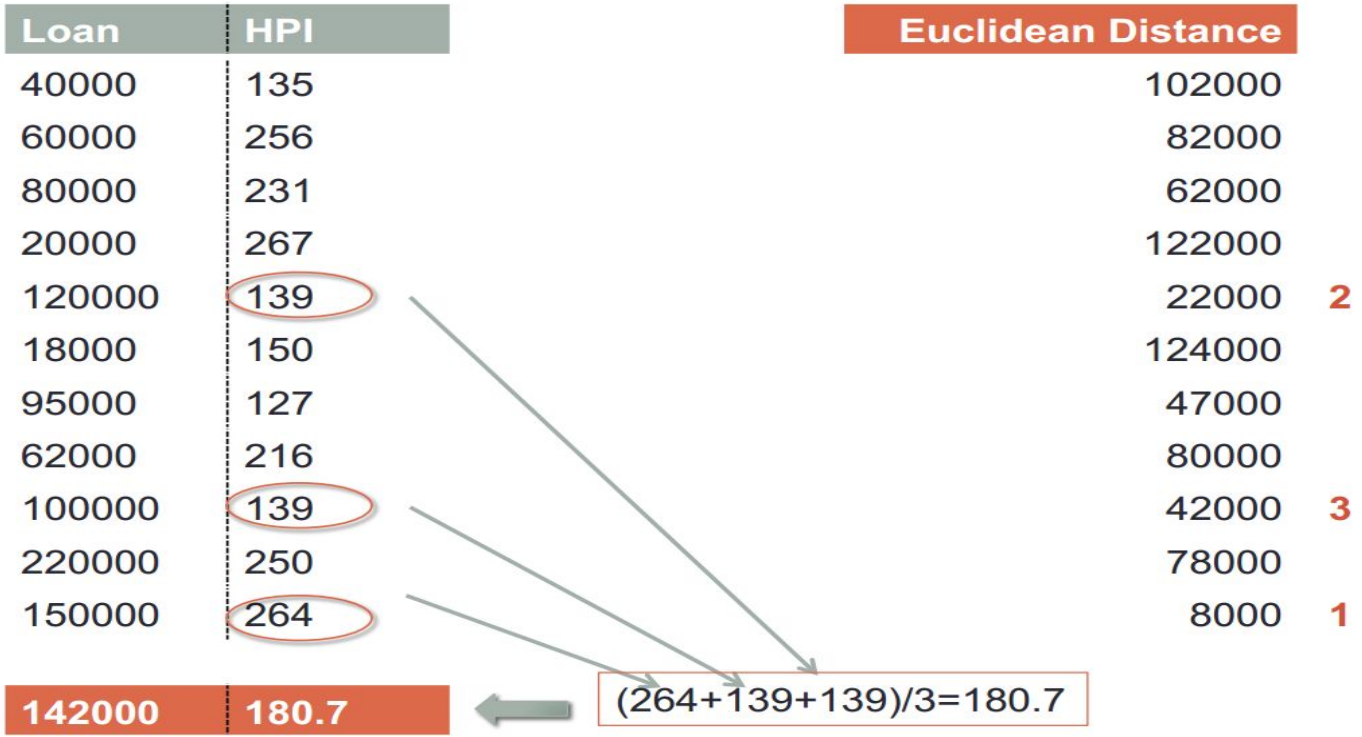
(3-NN) Regressor House Price Index (HPI)

- **Example**

Loan	HPI	Euclidean Distance	
40000	135	102000	
60000	256	82000	
80000	231	62000	
20000	267	122000	
120000	139	22000	2
18000	150	124000	
95000	127	47000	
62000	216	80000	
100000	139	42000	3
220000	250	78000	
150000	264	8000	1
142000	?		

(3-NN) Regressor House Price Index (HPI)

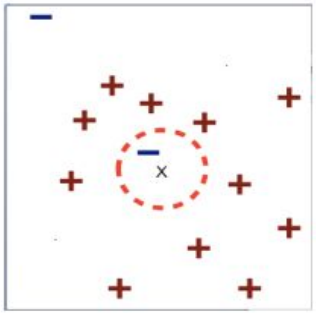
- **Example**

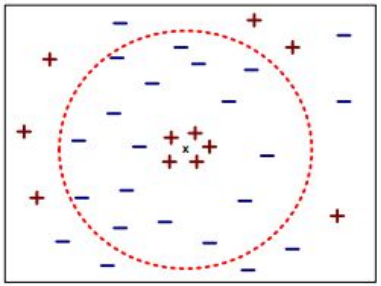


k selection

- A very small k may be sensitive to noise!.
- A very big k may introduce points of different class!.
- For k=n, all new points will be assigned the most frequent class!.

Model parameter tuning:
Use cross validation to tune the model's parameters (here k).



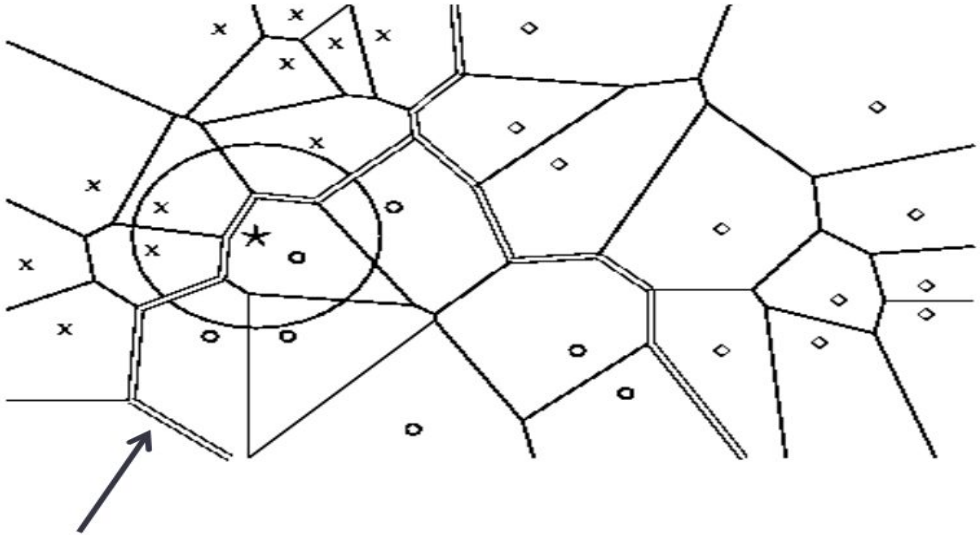


Remarks on k-NN classification

- The distance weighted k-NN algorithm appears to be robust to noisy training data and effective when provided with sufficiently large set of training examples.
- k-NN is a lazy learner.
- This is a local method, approximating the underlying target function locally, and this can help in smoothing out the impact of isolated noisy training examples. There is one problem, however, which is linked to the way the distance is calculated.
- When number of training examples is very large, k-NN method approaches the Bayesian optimal classification.

Linear and non Linear Classifiers

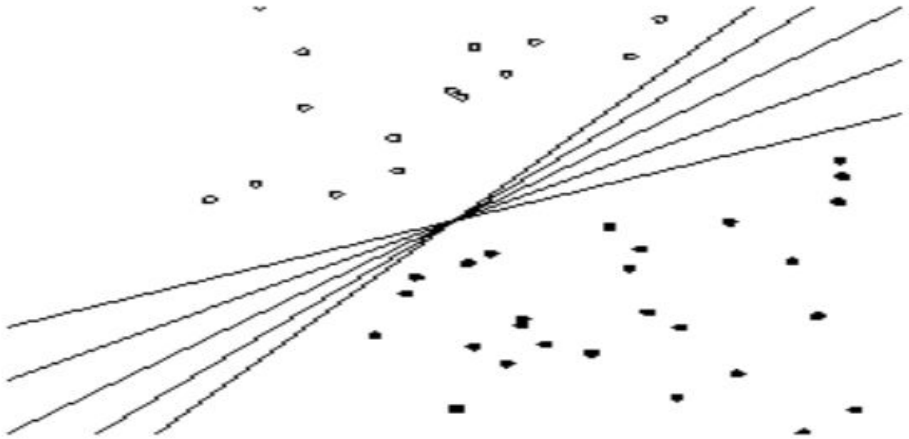
In K-NN we saw an example of a **non-linear classifier**: the decision boundary is not a straight line.



Decision boundary: Line that separates the three classes (x, circle, diamond) from each other.

Linear and non Linear Classifiers

On the contrary, for **linear** classifier the decision boundary is a **straight line (or a hyperplane)**.



Decision boundaries: Straight lines that divide points.

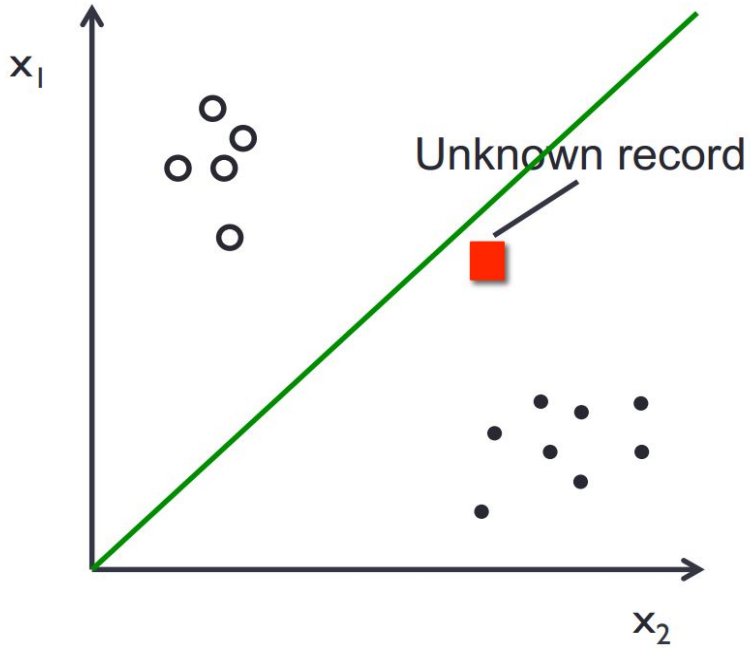
Classification problem

- **Problem:** Based on a labeled set of data points (**X,**Y) learn a function f: **X'** →Y in order to predict the label (class) of new unseen data points **X'.**
- The input feature vectors **x** may be **numeric**, but the target variable **y** is **categorical**.
- **Binary classification**: the target variable **y** can have only 2 values, representing the possible existing classes.
- **Multiclass classification**: the target variable **y** can have more than 2 values (but a finite number of them).

Linear Classifier

What is the class of the bullet?

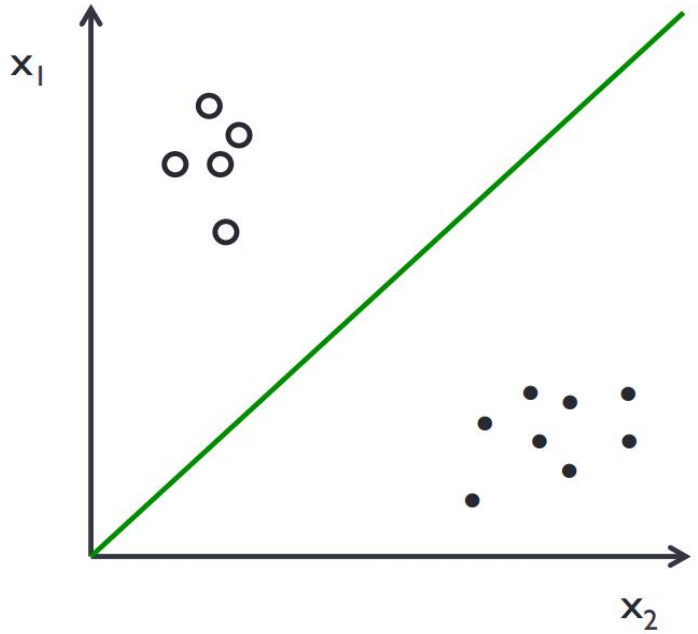
The line will tell us!



Linear Classifier

Linearly separable sets:

There exist a hyperplane (here a line) that correctly classifies all points.



Binary classification

Two classes to choose from – yes or no, negative or positive.

- Input:
  - training data: n dimensional vectors x (or points)
- Process:
  - Find a linear function described by the following equation:

$$f(\mathbf{x}) = w_1 x_1 + \ldots + w_n x_n + w_0 = \mathbf{w}^T \mathbf{x}$$

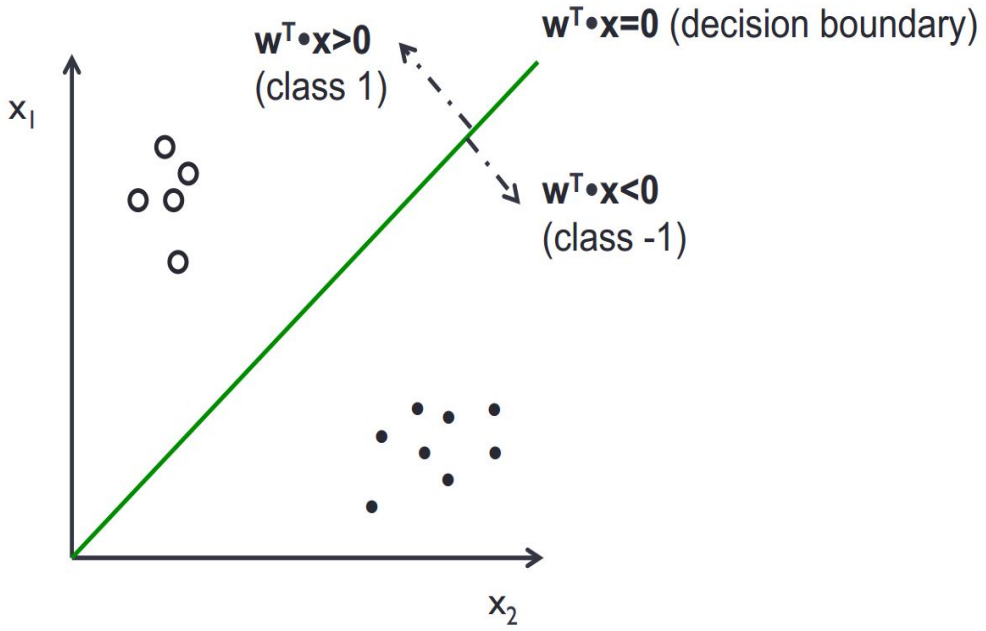
- The sign (+ or -) of f(x) will show the class of a newly given point. So, the predicted class **y** of a new point **x'** is:

$$y \begin{cases} = 1 & \text{if } f(\mathbf{x}') > 0 \\ = -1 & \text{if } f(\mathbf{x}') < 0 \end{cases}$$

- Para f(x) = 0 we get the boundary hyperplane.

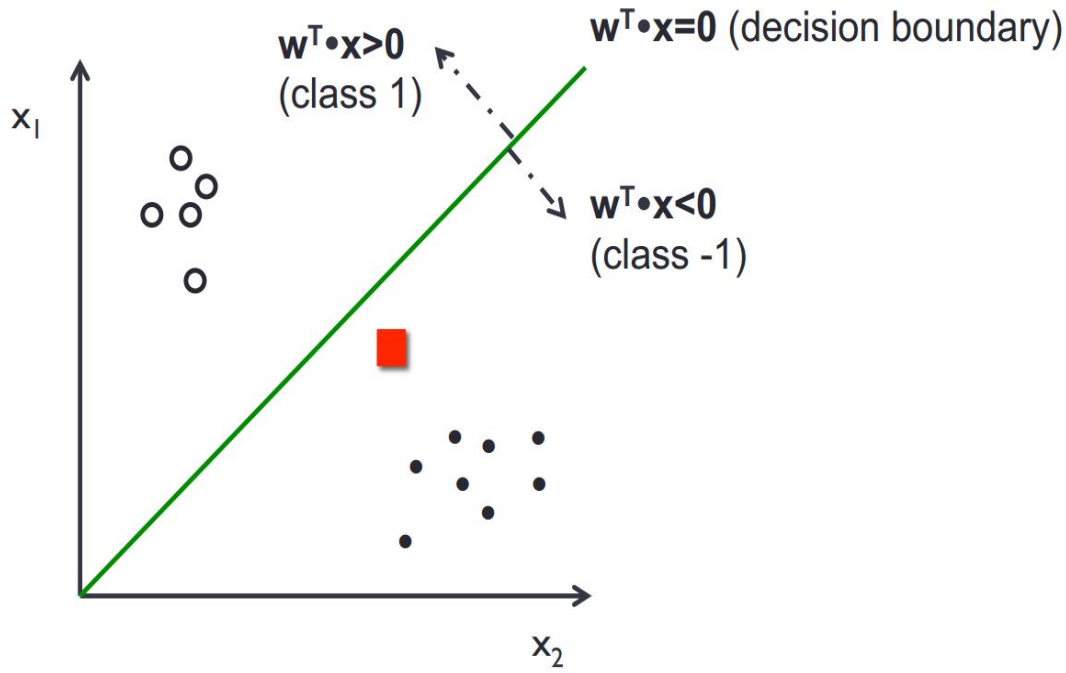
Binary Classification

There exist a hyperplane (here a line) that correctly classifies all points.



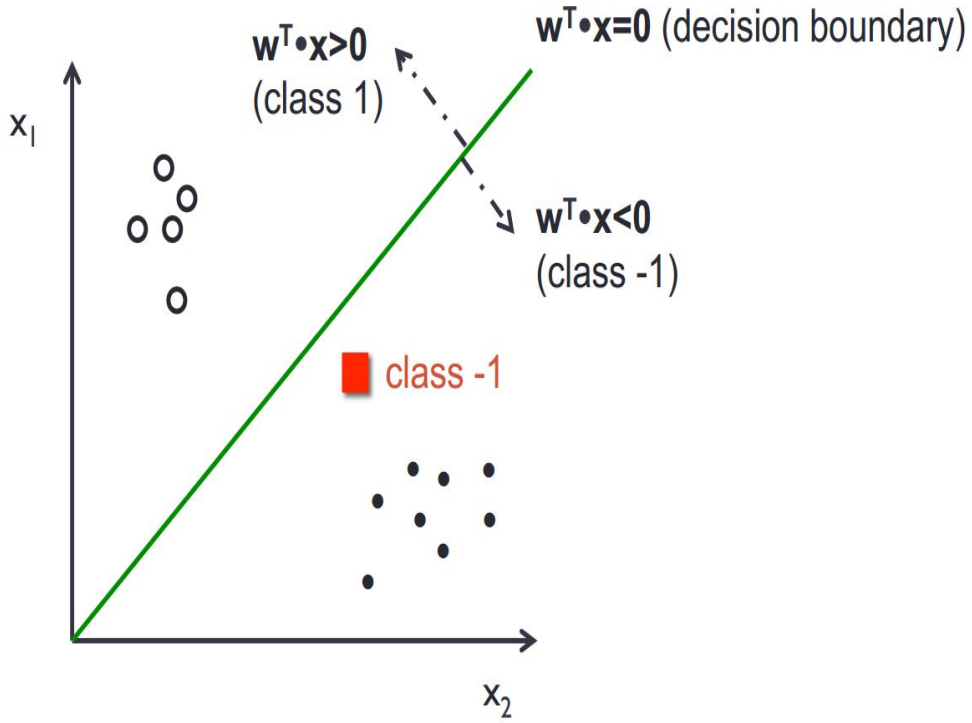
Binary Classification

There exist a hyperplane (here a line) that correctly classifies all points.The red square is in one of the two regions defined by the decision boundary.



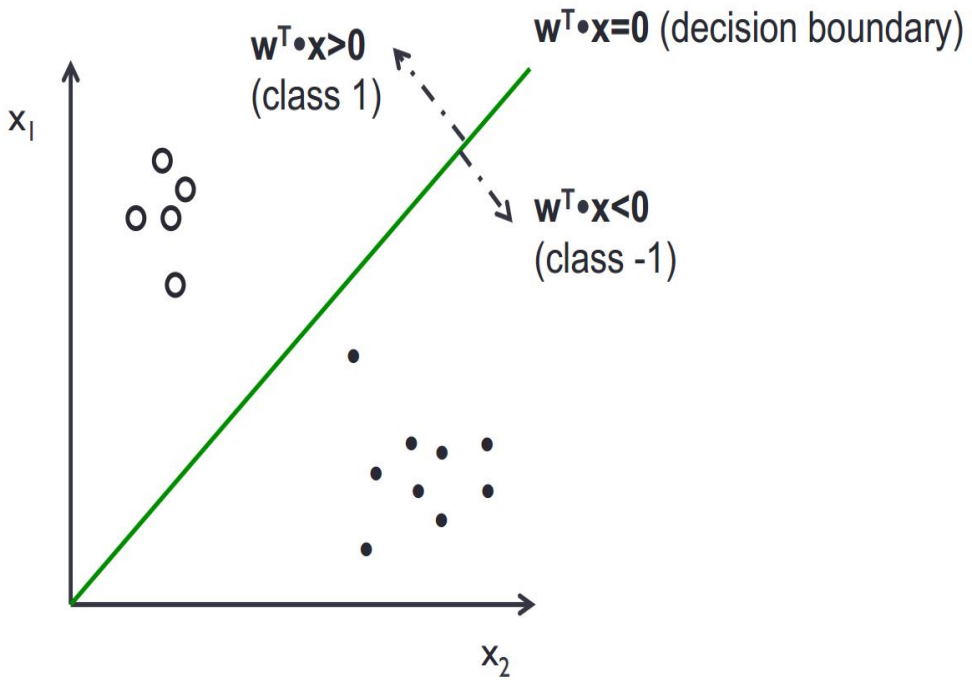
Binary Classification

There exist a hyperplane (here a line) that correctly classifies all points.The red square is in one of the two regions defined by the decision boundary. In this case: **class -1**.



Binary Classification

There exist a hyperplane (here a line) that correctly classifies all points. There are two regions that are formed from the decision boundary.



Learning the classification boundary

There are many algorithms that differ on:

- How they measure if the model fits well the training data (typically called the loss function).
- If they use regularization or not.

There are several algorithms related to this type of classification:

- Logistic Regression
- Naive Bayes
- SVC

Logistic Regression

- Despite its name, it is a classification method.
- It applies a sigmoid function:

$$\sigma(f(x)) = \frac{1}{1+e^{-f(x)}}$$ over a linear classifier $f(x)=\mathbf{w}^T\mathbf{x}$

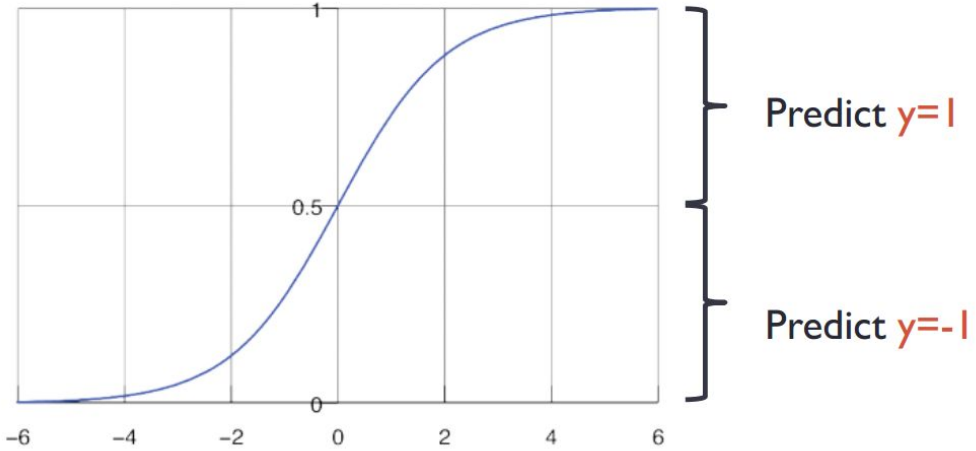
So, far a point **X**, we have two cases:

$$\sigma(f(\mathbf{x}))\begin{cases} \geq 0.5 & \text{then } \hat{y}=1 \\ < 0.5 & \text{then } \hat{y}=-1 \end{cases}$$

Logistic Regression

- It can also be interpreted as the **confidence** (probability) with which an element is in a binary (1,-1) class y.

$$p(y \mid \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}}$$



Obviously, $p(y = 1 \mid \mathbf{x}) = 1 - p(y = -1 \mid \mathbf{x})$

Logistic Regression

- Logistic regression is **generalized linear** because the decision boundary it produces is linear in function of **x.**
- To see this, consider the probabilities with which a certain point **x** belongs to the class y=1 and to the class y=−1.

$$\frac{p(y=1\,|\,\mathbf{x})}{p(y=-1\,|\,\mathbf{x})} = \frac{\dfrac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}}{1-\dfrac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}} = \ldots = e^{\mathbf{w}^T\mathbf{x}} \overset{\log}{\Leftrightarrow} \log\frac{p(y=1\,|\,\mathbf{x})}{p(y=-1\,|\,\mathbf{x})} = \mathbf{w}^T\mathbf{x}$$

- For the decision boundary the probabilities are equal to 0.5, so

$$\log\frac{0.5}{0.5} = 0 = \mathbf{w}^T\mathbf{x}$$

Decision boundary hyperplane

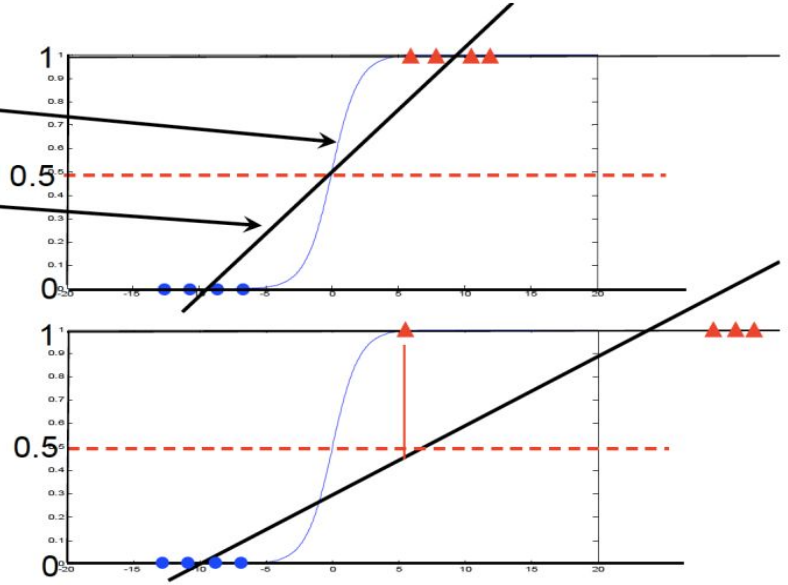
Logistic Regression

Why do we choose the sigmoid function to fit the data?

Least squares fit

$\sigma(w_1 \mathbf{x} + w_0)$ fit to y

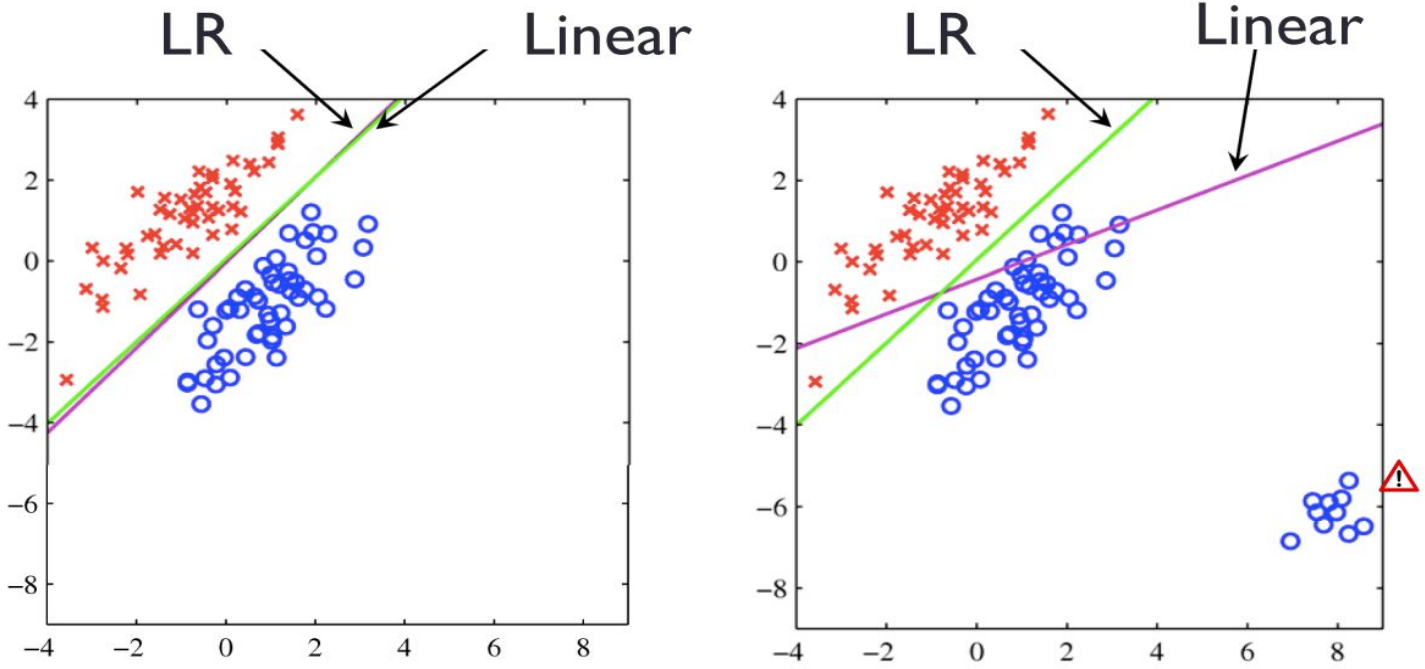
$w_1 \mathbf{x} + w_0$ fit to y



- *Fit of $w_1 \mathbf{x} + w_0$ is dominated by the more distant points and ..*
- *causes misclassification.*
- *Instead LR regresses the sigmoid to the class data.*

Logistic Regression

Similarly in the 2D space



Logistic Regression

To compute the parameters **w** we have to solve an **optimization problem**. For this we are going to use the data **likelihood function:**

$$l(\mathbf{w}) = \prod_{i=1}^{N} \frac{1}{1 + e^{-y_i \mathbf{w}^T x_i}}$$

We search for **w** that maximizes the likelihood of the data:

$$\log \prod_{i=1}^{N} \frac{1}{1 + e^{-y_i \mathbf{w}^T x_i}} = \sum_{i=1}^{N} \log\left(\frac{1}{1 + e^{-y_i \mathbf{w}^T x_i}}\right) = -\sum_{i=1}^{N} \log\left(1 + e^{-y_i \mathbf{w}^T x_i}\right)$$

For convenience, the **loss function** is the negative logarithm of the data likelihood:

$$L(\mathbf{w}) = \sum_{i=1}^{N} \log\left(1 + e^{-y_i \mathbf{w}^T x_i}\right)$$

Logistic Regression

Finally, w is the argument that minimizes L(**w**):

$$\mathbf{w} = \text{argmin}\, L(\mathbf{w}) = \text{argmin} \sum_{i-1}^{N} \log\left(1 + e^{-y_i \mathbf{w}^T x_i}\right)$$

As in linear regression, in logistic regression we can add a **regularization term r(w)** (L_1 or L_2):

$$\mathbf{w} = \text{argmin}\left(\sum_{i-1}^{N} \log\left(1 + e^{-y_i \mathbf{w}^T x_i}\right) + Cr(\mathbf{w})\right)$$

where C is the tuning parameter and r(w) is $||\mathbf{w}||_2^2$ or $||\mathbf{w}||_1$.

Note that via regularization, we also allow for small misclassifications.

Logistic Regression

$$\mathbf{w} = \text{argmin}\left(\sum_{i-1}^{N} \log\left(1 + e^{-y_i \mathbf{w}^T x_i}\right) + Cr(\mathbf{w})\right)$$

- For correctly classified points $-y_i \mathbf{w}^T(\mathbf{x}_i)$ is negative, and thus $\log(1 + e^{-y_i \mathbf{w}^T(\mathbf{x}_i)})$ is near zero.
- For incorrectly classified points $-y_i \mathbf{w}^T(\mathbf{x}_i)$ is positive, and thus $\log(1 + e^{-y_i \mathbf{w}^T(\mathbf{x}_i)})$ can be large.
- Hence the regularization penalizes parameters which lead to such misclassifications.

Naive Bayes Classifier

Probability Basics:

Prior, conditional and joint probability for random variables.

- Prior probability: $P(x)$
- Conditional probability: $P(x_1 \mid x_2), P(x_2 \mid x_1)$
- Joint probability: $\mathbf{x} = (x_1, x_2), P(\mathbf{x}) = P(x_1, x_2)$
- Relationship: $P(x_1, x_2) = P(x_2 \mid x_1)P(x_1) = P(x_1 \mid x_2)P(x_2)$
- Independence:

$$P(x_2 \mid x_1) = P(x_2), P(x_1 \mid x_2) = P(x_1), P(x_1, x_2) = P(x_1)P(x_2)$$

Naive Bayes Classifier

- Assumes that the features **x** are **conditionally independent,** given the output **y.**
- Here we will see the method for **discrete values**.
- Uses the **Bayesian rule**.

$$P(y \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid y)P(y)}{P(\mathbf{x})}$$

$$Posterior = \frac{Likelihood \times Prior}{Evidence}$$

Naive Bayes Classifier

From the last equation, we describe each of its components:

- $P(y|\mathbf{x})$: the posterior probability of class (target), given an attribute vector (predictor).
- $P(y)$: the prior class probability
- $P(\mathbf{x}|y)$: the likelihood, i.e., the probability of generating the predictor $\mathbf{x}$ given target y
- $P(\mathbf{x})$: the evidence, i.e., the prior probability of the predictor $\mathbf{x}$

Naive Bayes Classifier

Maximum A Posterior (MAP) Classification Rule.

To classify an input **x:**

- find the posterior probabilities $P(y_i|\mathbf{x})$ for each output class y_i.
- Assign the label y to **x** if $P(y|\mathbf{x})$ is the highest among all $P(y_i|\mathbf{x})$s.

Note that since the factor P(x) in the Bayesian rule is common for all $P(y_i|\mathbf{x})$, we can omit it when applying MAP.

$$P(y_i / \mathbf{x}) = \frac{P(\mathbf{x} / y_i)P(y_i)}{P(\mathbf{x})} \propto P(\mathbf{x} / y_i)P(y_i)$$

Naive Bayes Classifier

Maximum A Posterior (MAP) Classification Rule.

So, the classification rule can be written as:

$$\hat{y} = \underset{y}{\operatorname{argmax}} P(y) \prod_{i=1}^{n} P(x_i / y)$$

Naives Bayes classifier has a fast training phase, even with many features because it looks and collects statistics from each feature individually.

Naive Bayes Classifier as a linear classifier

Naive Bayes Classifier can be seen as a **linear classifier** if we consider the **log version** of the classification rule

$$f(\mathbf{x}) = \log \frac{P(y=1/\mathbf{x})}{P(y=0/\mathbf{x})} = \log P(y=1/\mathbf{x}) - \log P(y=0/\mathbf{x})$$

$$= (\log \theta_1 - \log \theta_0)^{\mathrm{T}} \mathbf{x} + (\log P(y=1) - \log P(y=0))$$

f(**x**) is a linear function in **x**. f(**x**)=0 is the decision boundary.

Naive Bayes Classifier

Example: Play Tennis depending on weather conditions

PlayTennis: training examples

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Is it probable that we will play tennis on a (Sunny, Hot, Normal, Strong) day?

Naive Bayes Classifier

Step 1: Build **frequency** table for each attribute

Outlook	Play=*Yes*	Play=*No*
Sunny	2	3
Overcast	4	0
Rain	3	2

Temperature	Play=*Yes*	Play=*No*
Hot	2	2
Mild	4	2
Cool	3	1

Humidity	Play=*Yes*	Play=*No*
High	3	4
Normal	6	1

Wind	Play=*Yes*	Play=*No*
Strong	3	3
Weak	6	2

Naive Bayes Classifier

Step 2: Build **likelihood** table for each attribute

Outlook	Play=*Yes*	Play=*No*
Sunny	2/9	3/5
Overcast	4/9	0/5
Rain	3/9	2/5

Temperature	Play=*Yes*	Play=*No*
Hot	2/9	2/5
Mild	4/9	2/5
Cool	3/9	1/5

Humidity	Play=*Yes*	Play=No
High	3/9	4/5
Normal	6/9	1/5

	9/14	*5/14*

Wind	Play=*Yes*	Play=*No*
Strong	3/9	3/5
Weak	6/9	2/5

Naive Bayes Classifier

Step 3: Compute the probabilities

Outlook	Play=*Yes*	Play=*No*
Sunny	2/9	3/5
Overcast	4/9	0/5
Rain	3/9	2/5

Temperature	Play=*Yes*	Play=*No*
Hot	2/9	2/5
Mild	4/9	2/5
Cool	3/9	1/5

$P(x \mid y) = P(Strong \mid yes) = 3/9 = 0.33$

Humidity	Play=*Yes*	Play=*No*
High	3/9	4/5
Normal	6/9	1/5

Wind	Play=*Yes*	Play=*No*
Strong	3/9	3/5
Weak	6/9	2/5

	9/14	*5/14*

$P(y) = P(yes) = 9/14 = 0.64$

$P(x) = P(Strong) = 6/14 = 0.43$

$P(y \mid x) = P(yes \mid Strong) = 0.33 * 0.64 / 0.43 = 0.49$

Naive Bayes Classifier

Question: Is it probable that we will play tennis on a **x**=(Outlook:Sunny, Temp:Hot, Humidity:Normal,Wind:Strong) day?
Answer: Let's apply **MAP** for P(yes, **x**) and P(no, **x**).

- $Likelihood_{yes}=P(\mathbf{x}|yes)=P(Outlook=Sunny|yes)*P(Temp=Hot|yes)*P(Humidity=Normal|yes)*P(Wind=Strong|yes)=2/9*2/9*6/9*3/9=0.043$
- $Likelihood_{no}=P(\mathbf{x}|no)=P(Outlook=Sunny|no)*P(Temp=Hot|no)*P(Humidity=Normal|no)*P(Wind=Strong|no)=3/5*2/5*1/5*3/5=0.029$
- P(yes)=9/14=0.64
- P(no)=5/14=0.36

- P(yes|**x**)=P(**x**|yes)*P(yes)=0.043*0.64=0.03
- P(no|**x**)=P(**x**|no)*P(no)=0.029*0.36=0.01

So, since P(yes|**x**)>P(no|**x**) then **x** is classified as a yes.

Naive Bayes Classifier

The **zero-frequency problem**:
When an attribute has zero frequency for a yi, then the likelihood $P(x|y_i)$ will be equal to zero!.
To avoid this, we can simply augment all the counts by one. By example:

Outlook	Play=*Yes*	Play=*No*
Sunny	2	3
Overcast	4	0
Rain	3	2

Outlook	Play=*Yes*	Play=*No*
Sunny	3	4
Overcast	5	1
Rain	4	3

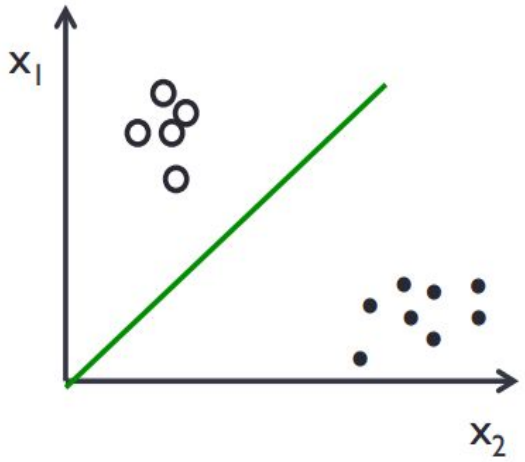
Linear SVC

The objective of a **Linear SVC** (Support Vector Classifier) is to fit to the data you provide, returning a "best fit" hyperplane that divides, or categorizes, your data.

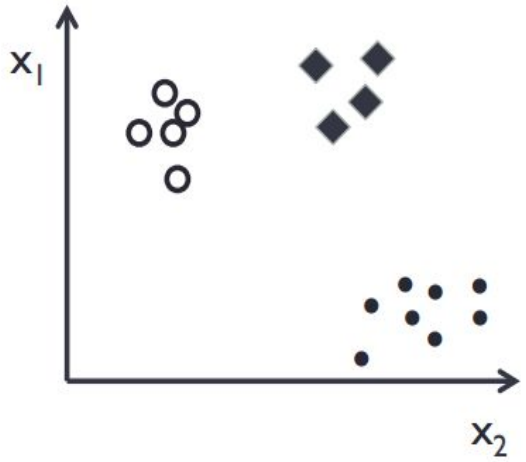
From there, after getting the hyperplane, you can then feed some features to your classifier to see what the "predicted" class is. This makes this specific algorithm rather suitable for our uses, though you can use this for many situations.

According to sklearn documentation, the algorithm used in LinearSVC is much more efficient and can scale almost linearly to millions of samples and/or features.

Binary vs multiclass classifiers



Binary

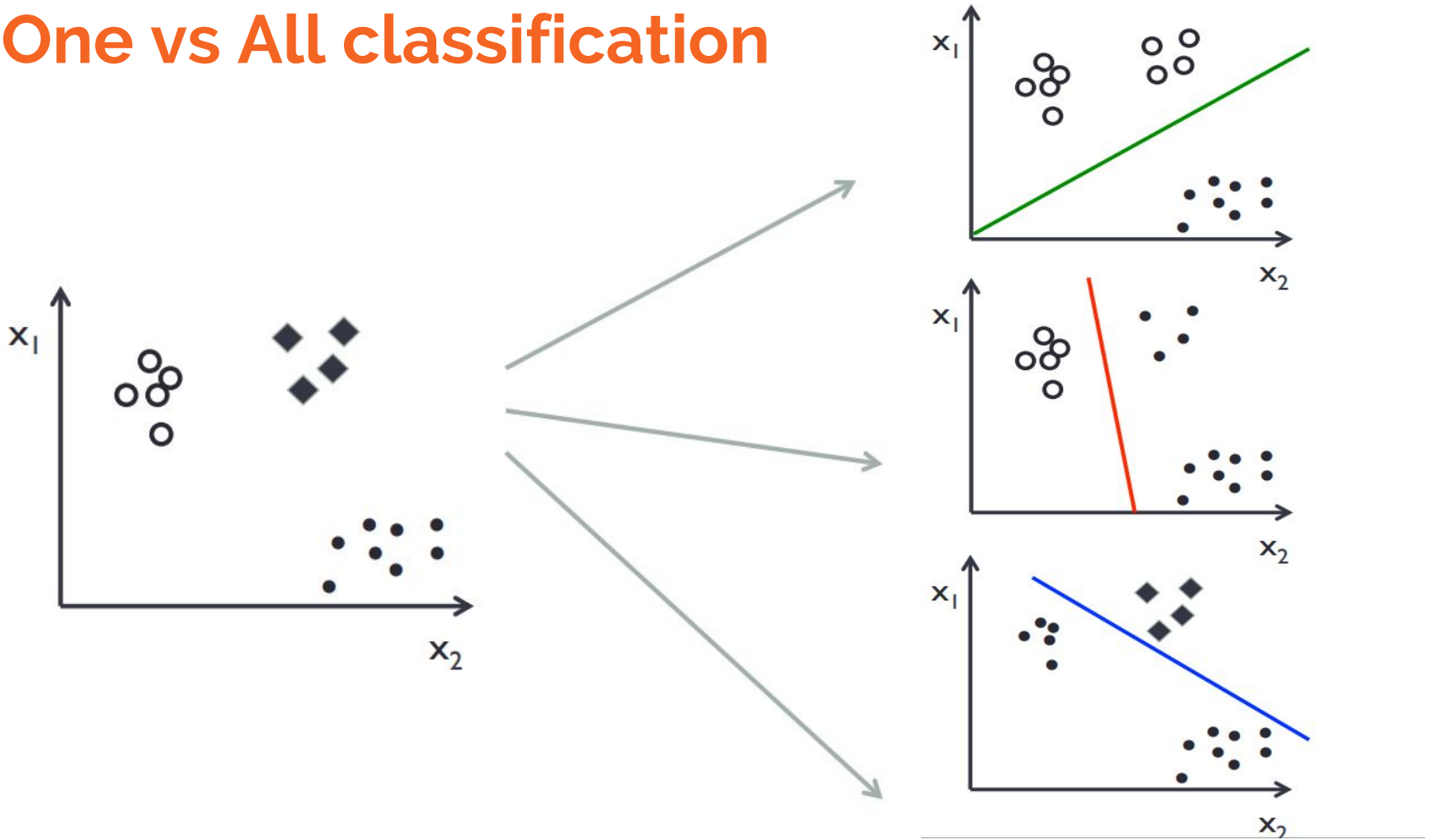


Multiclass

Linear models for multiclass classification

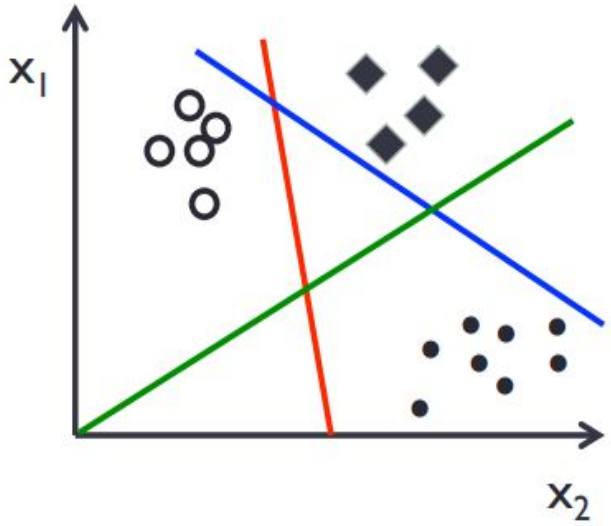
- Multiclass classification: each sample can have one class out of many possibilities.
- Linear algorithms (LR, SVC, Naive Bayes) that we saw can be used as per se, also for multiclass problems. How? **One-vs-All.**
- **One-vs-All**: Binary problem with class 0 being some class and class 1 being the rest of the classes.
  - If the set of possible classes is Y then, for each $y_i \in Y$, learn a line separating class y_i from the rest of the classes using a binary classifier.
  - Use all the learned lines to separate the classes.

One vs All classification



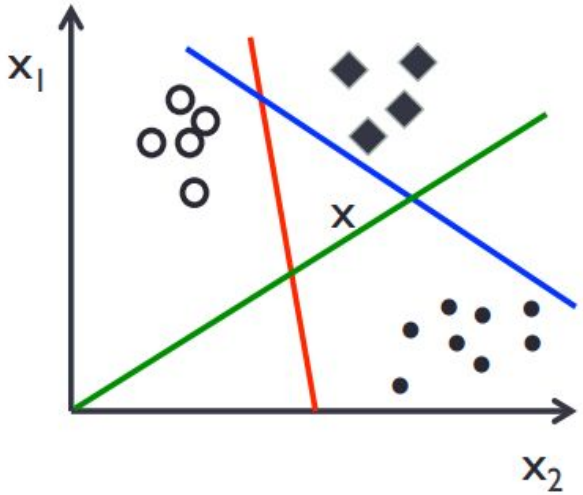
One vs All classification

Do you see any problem here?



One vs All classification

What about the points in the triangle?
Choose the line closest to the point!. The class with the highest probability.



Eg for LR we compute:

$p(y_i \mid \mathbf{x})$ for y_i in {o, •, ◆}

The class with the highest probability for a new point **x** is its predicted class.

Exercises

Choose one of the pre-established data by scikit learn and execute the steps proposed by your API with the following models.

- Linear Regression
- Ridge Regression
- Lasso Regression
- Logistic Regression
- Gaussian Naive Bayes

Sources

- This presentation is based on the work of Katerina Tzompanaki: https://perso-etis.ensea.fr//tzompanaki/.