UNIVERSITÀ DI PARMA

DIPARTIMENTO DI INGEGNERIA E ARCHITETTURA

Classification

Gianfranco Lombardo, Ph.D
gianfranco.lombardo@unipr.it

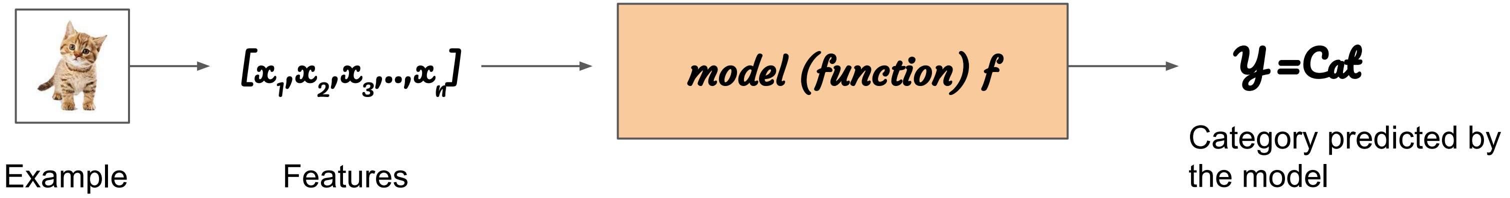
Review: What do we mean by learning?

- "A computer program is said to learn from ***experience E*** with respect to some class of ***tasks T*** and ***performance measure P***,
  if its performance at tasks in T, as measured by P, improves with experience E".
  (Mitchell 1997)
- Learning is our means of attaining the ability to perform automatically a task
- **Task T** : A task that is difficult to be solved with fixed programs written and designed by human beings
- **Experience E**: Collected data that describes the input of our ML system and the main source of information to exploit in order to learn
- **Performance measure P**: How good is the model? Is it able to solve the problem for real? Obviously depending on the task we have to choose a different measure

Today: Classification

- **Classification**: The system is asked to specify which of k categories some input belongs to. For example:
    - Given a sentence (maybe a tweet) the system should determines if it express a positive or negative or neutral feeling (K=3)
    - Given an image where it can be a dog or a cat, we want to determine with the system which one is present (K=2)

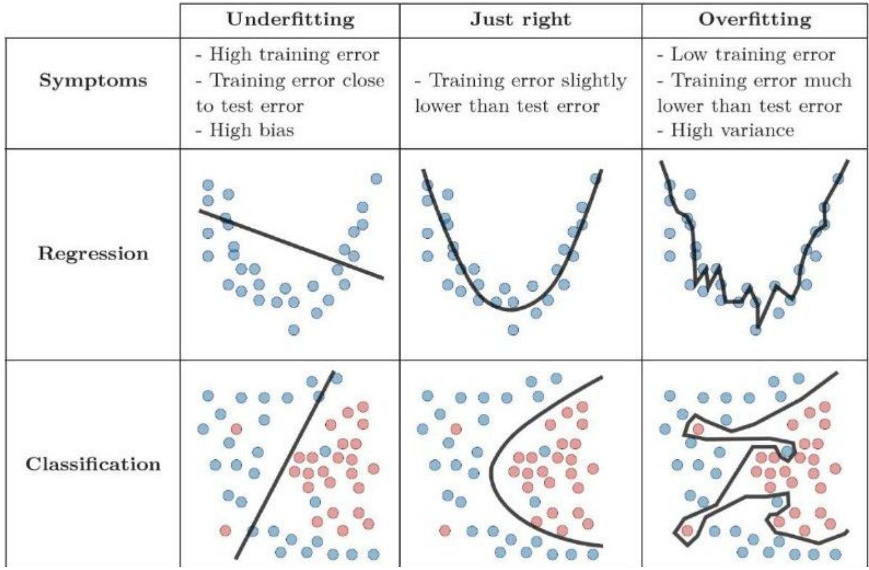
- To solve this task, the learning algorithm is usually asked to produce a function $y = f(x): \mathcal{R}^n \rightarrow \{1,\ldots,k\}$
    - So the model takes an example **x** as input and after some processing **f(x)** it returns a value **y** that is one of the k categories the example x should belong to.



Model for classification task

- We want to estimate a parametric function $\mathcal{F}(\mathcal{X},\mathcal{W})$ that maps our input features $\mathcal{X}$ to one or more of our target labels in y, where y is an element of a limited set and is called class.

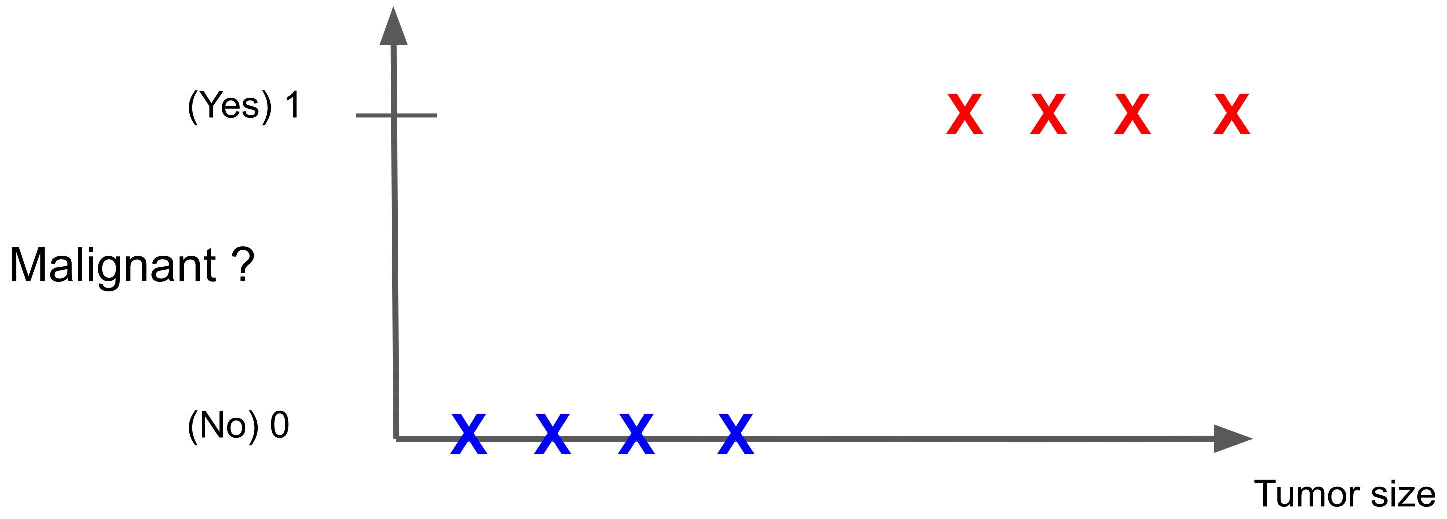
- The goal is to find the **decision boundaries** in the features space



	Underfitting	Just right	Overfitting
Symptoms	- High training error - Training error close to test error - High bias	- Training error slightly lower than test error	- Low training error - Training error much lower than test error - High variance
Regression			
Classification			

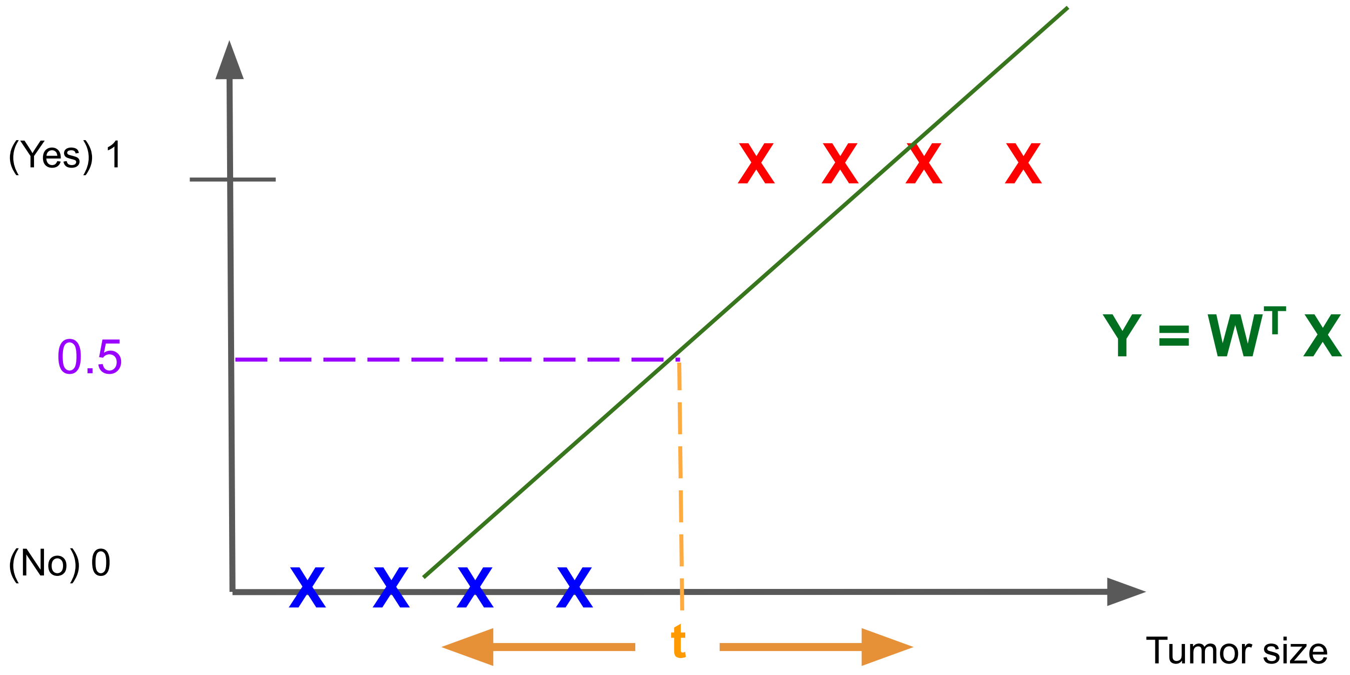
Can we adapt Linear Regression for a classification task?

- We want to train a model to predict if a tumor is benign or malignant using a single feature (tumor size)
- $y \in \{0,1\}$: 0 is malignant , 1 is benign
- X has only one feature



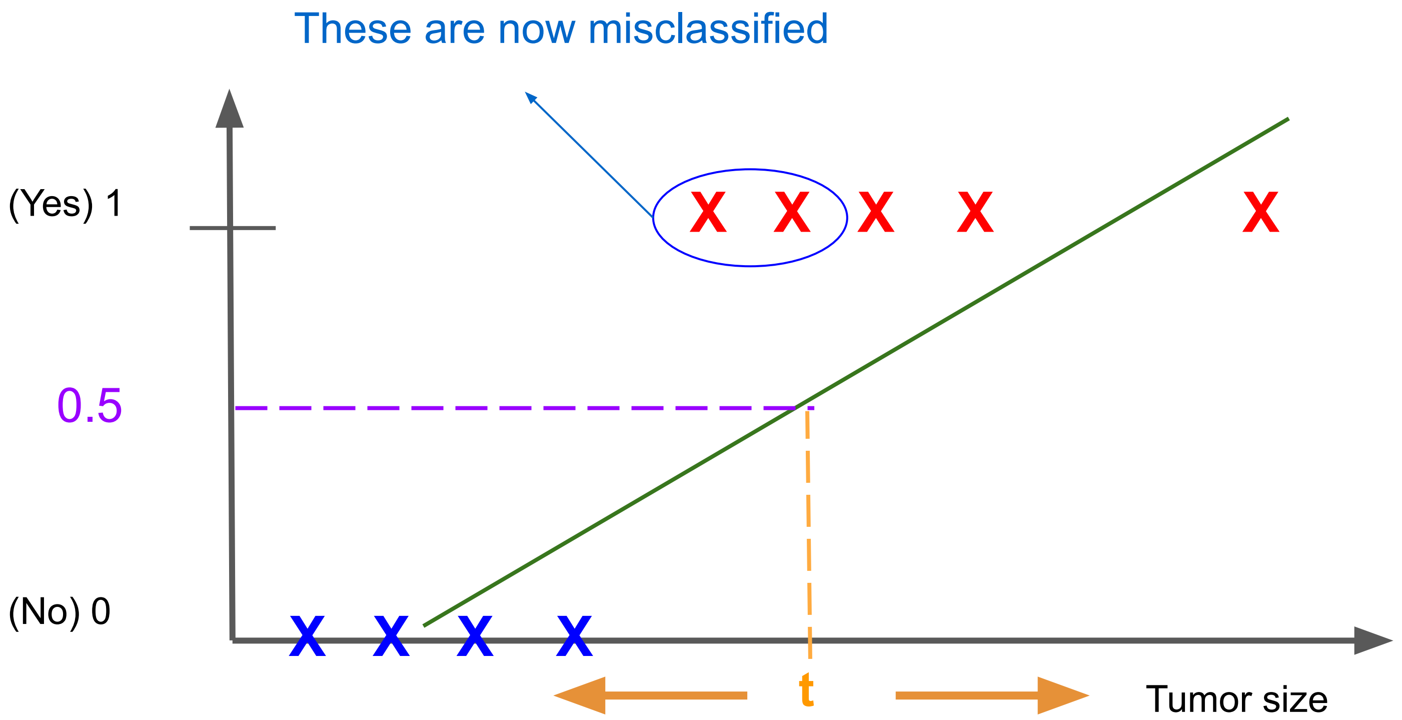
Can we adapt Linear Regression for a classification task?

- We can find the best $\mathbf{Y = W^T X}$ from our dataset and then select a threshold over y to classify when the tumor is malignant or not



Can we adapt Linear Regression for a classification task?

- Does it really work? Or we were just lucky? What happens if i add another training example ?



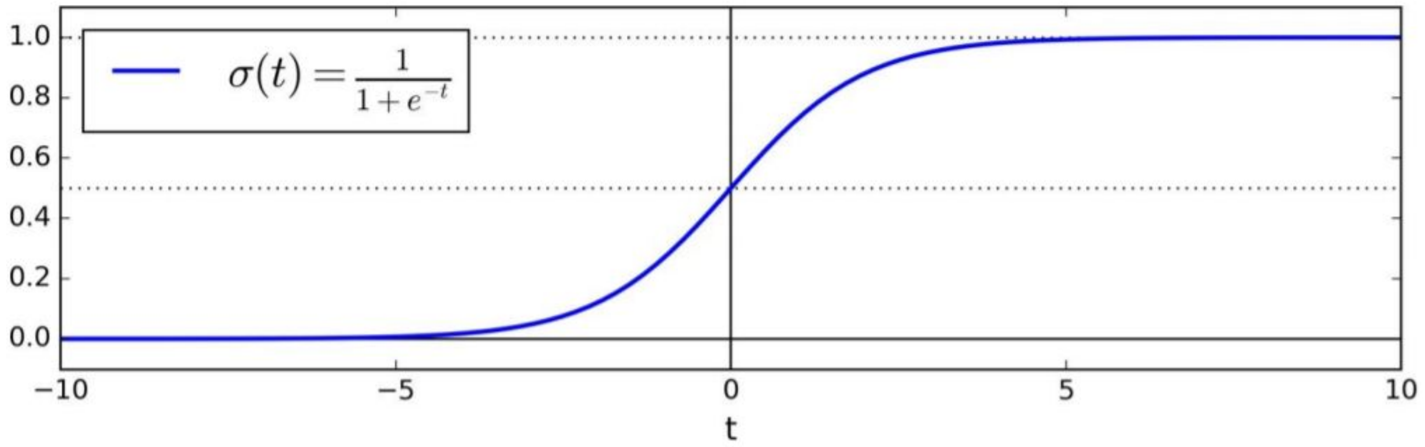
Logistic regression

- Moreover, linear regression returns a y that is not bounded between 0 and 1
- We want a $h_W(X)$ bounded: $0 <= h_W(X) <= 1$, the simplicity of a linear regression and a good fit to our binary classification task
    - $y = h_W(X) = \sigma(W^T X)$
- Logistic regression (sigmoid) can be the perfect solution

$$h_W(\mathrm{x}) = \frac{1}{1+e^{-W^T x}}$$

Logistic function

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$



Logistic Regression

- Logistic regression (despite the name) is a classification algorithm
- The name is due to the past when the logit function was invented and used as a regression algorithm
- As a classification algorithm it estimates the probability that an instance belongs to a class or not (Binary classification).
    - If the estimated probability is greater than 50% then the model predicts that class

- The logistic function is a sigmoid function (S-shape) that outputs a number between 0 and 1

$$h_W(\mathrm{x}) = \frac{1}{1+e^{-W^T x}}$$

Logistic function

- Just like a linear regression model, a logistic regression computes a weighted sum of the input features plus a bias term

- But instead of returning the result directly, it outputs the logistic of this result
  - So when the probability that an instance belongs to the positive class is computed, the $\mathbf{y}^*$ can be computed easily:
    - Class 0 (or negative) if $p < 0.5$
    - Class 1 (or positive) if $p >= 0.5$

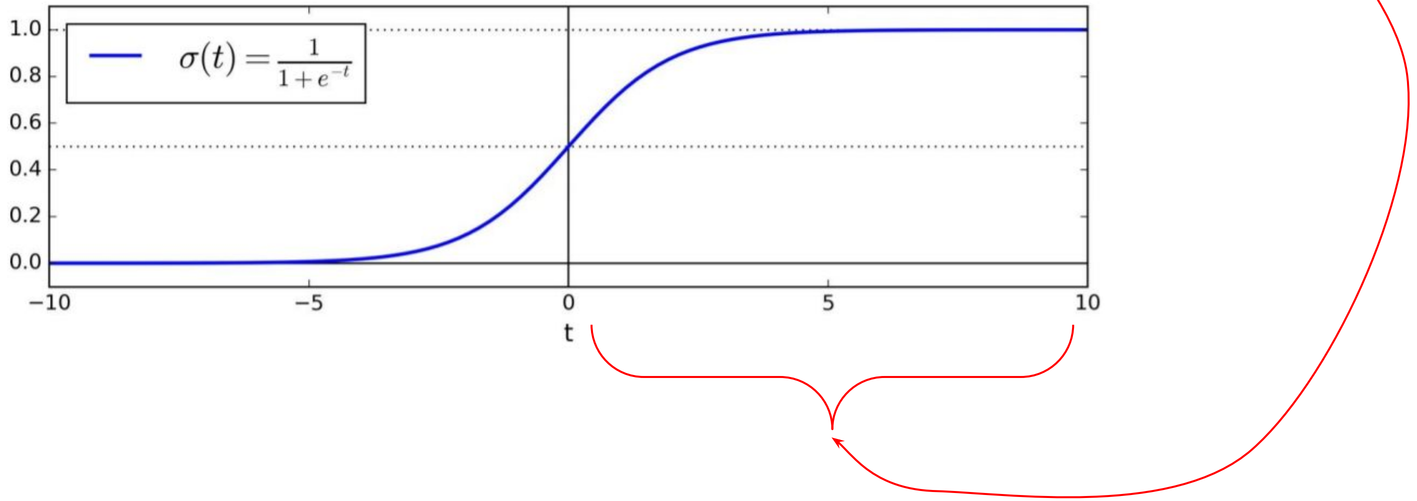
- When possible labels are more than two, several binary classifier are built to identify the correct class

Logistic function

- In our case, $h_W(X)$ with Logistic regression estimated the probability that y=1 on input X
- For example, If $h_W(X) = 0.7$ for a sample x , It means that patient has the probability of 70% of having a malignant tumor.
- If you are familiar with probability, we can see $h_W(X)$ in the following way:
  - $h_W(X) = P(y=1 \mid X; W)$
  - The probability that y=1 given X parametrized by W

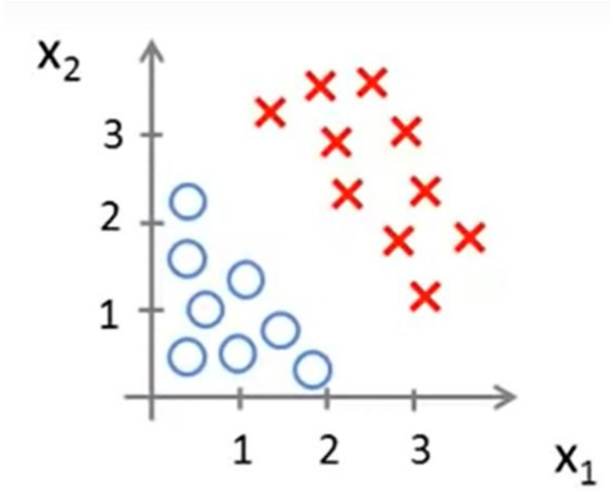
Decision boundaries

- We predict y=1 when $h_W(X)$ >= 0.5 that means when $W^TX >= 0$ and vice-versa for y=0
  - Remember y = $h_W(X) = \sigma(W^TX)$ *with* $\sigma(z) = \frac{1}{1+\exp(-z)}$



Decision boundaries

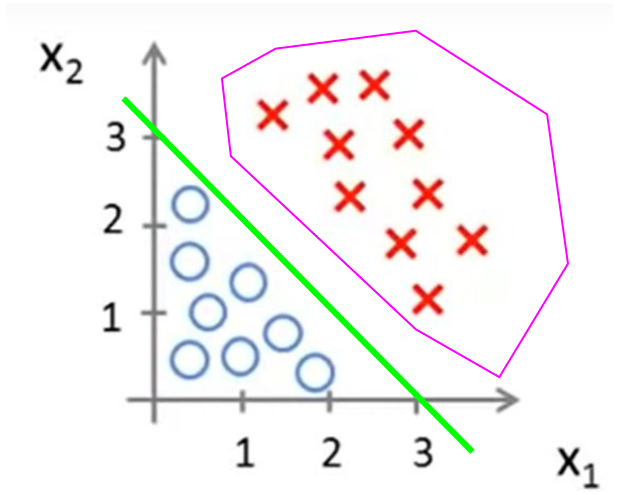
- Suppose the case we have the dataset in figure, with 2 features X_1 and X_2
  - We trained a Logistic regression and we found that the best parameters W are respectively $W_0 = -3$, $W_1=1$, $W_2=1$



$$h_W(X) = g(w_0 + w_1x_1 + w_2x_2)$$

Decision boundaries

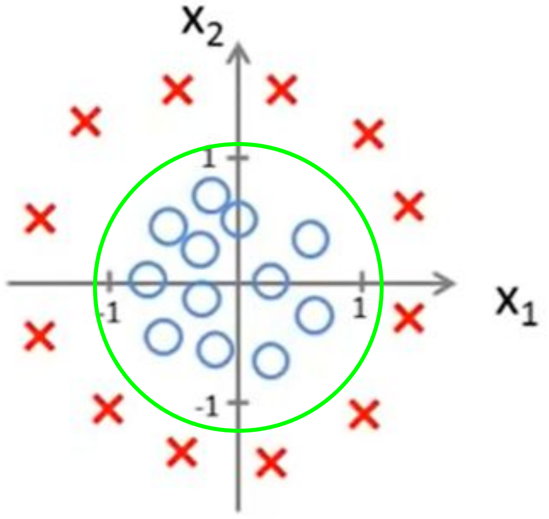
- We predict y=1 when $-3 + x_1 + x_2 >= 0$
  - So when **$x_1 + x_2 >= 3$**



The green line is an example of a decision boundary

Note: Decision boundary is not a property of the training set!
Training set must be used to fit the parameters w!

Non-linear decision boundaries



$$h_W(X) = g(w_0 + w_1x_1 + w_2x_2 +$$
$$w_3\, x^2{}_1 + w_4\, x^2{}_2\,)$$

Learning: An optimization problem (again)

Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$

m examples $\quad x \in \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix} \quad x_0 = 1, y \in \{0, 1\}$

$$h_W(\mathrm{x}) = \frac{1}{1+e^{-W^T x}}$$

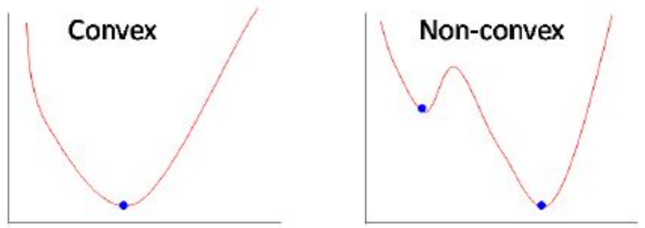
How to choose parameters W ?

Learning: An optimization problem (again)

$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(y_i - y_i^*)^2$$

$$h_W(x) = \frac{1}{1+e^{-W^T x}}$$

This cost function definition introduces a non linearity that makes the function we want to minimize non-convex



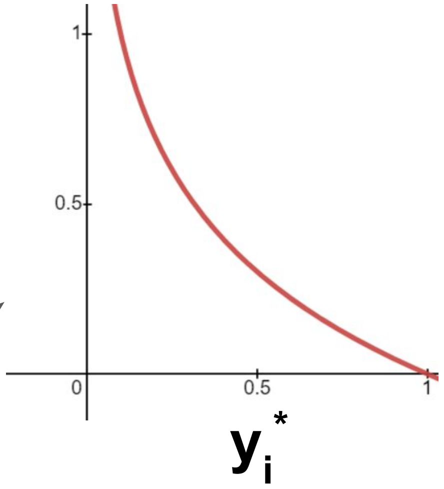
With a Non-convex function, there are not guarantees to converge in a global optimum

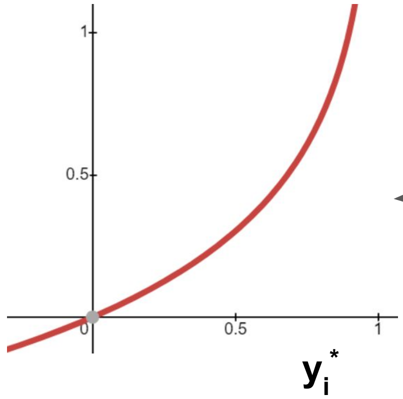
Learning: An optimization problem

Let's define the cost function in the following way to have a convex function

$$J(w) = \frac{1}{m} \sum_{i=1}^{m} Cost(W, y)$$

$$Cost(W, y) = \begin{cases} -\log(y_i^*) & if\ y = 1 \\ -\log(1 - y_i^*) & if\ y = 0 \end{cases}$$





Cost = 0 if y=1 and y^*=1
Cost -> ∞ if y=1 and y*= 0
We will penalize learning algorithm by a very large cost

Learning: An optimization problem

Merge Cost in a single function and apply Gradient Descent

$$J(w) = -\frac{1}{m}\left[\sum_{i=1}^{m} y_i \log(y_i^*) + (1-y_i)\log(1-y_i^*)\right]$$

Repeat until convergence {

$$w_J := w_j - \alpha \frac{d}{dw_j} J(W) =$$

$$\text{with } y_i^* = h_W(x) = \frac{1}{1+e^{-W^T x}}$$

$$= w_j - \alpha \sum_{i=1}^{m} (y_i^* - y_i)\, x_{j_{(i)}}$$

}

Multi-class classification

- Commonly the world is not binary and we want to classify our data in more than two classes!
    - For example: Iris dataset: Versicolor, Virginica, Setosa (3 classes!)
- In this case we say that the problem is **multi-class**

- Another case is when we want to predict for each sample more than one label and these are not mutually exclusive
    - Example Face Recognition
    - Example Topic detection: we want to classify technical documentation and each document can be associate to more than one label
        - A paper can be classified about its topic “Computer science” and then also with its subtopics “Artificial Intelligence”
    - Every sample can be associated to one or more labels
    - It is a more complex task and we won’t talk about it in this course but remember that is called a **multi-label task**

Multi-class classification: One-vs-ALL

- Example: We have three classes to be predicted: 🔴 🟩 ▲

- We train a Logistic classifier, but wait! Logistic classifier can predict only on a binary case!

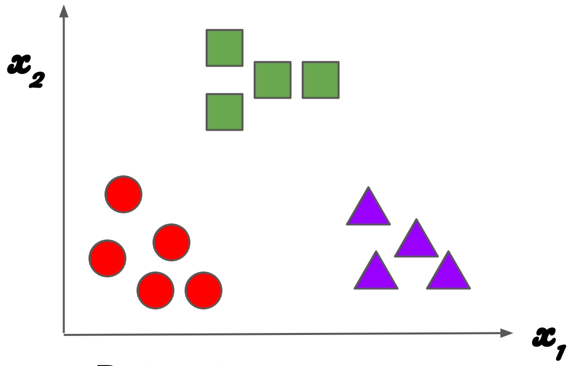
- The solution is the **One-vs-ALL** (One-vs-Rest) approach:
  - We train one classifier for each class that should be recognized. If three classes -> Three classifiers
  - The first one will recognize if the sample is a dot or other, the second one if the sample is a square or other, the third if the sample is a triangle or other.

- So we have a model $\boldsymbol{h_w^{(i)}(x) = P(y=i \mid x; w)}$ where i is the class, i=1,2,3

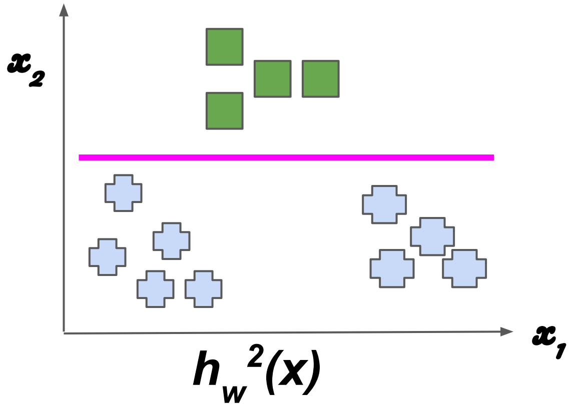
Multi-class classification: One-vs-ALL

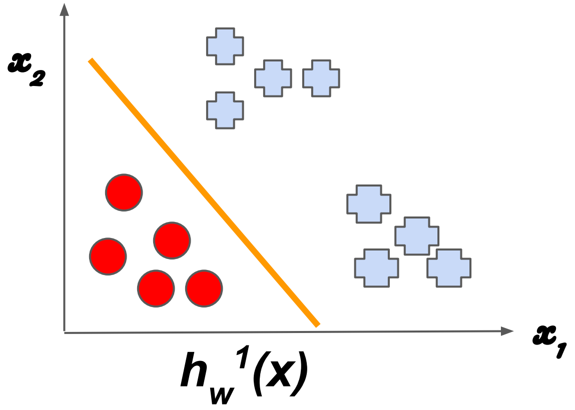
- Considering $\boldsymbol{h_w^{(i)}(x) = P(y=i \mid x; w)}$ where i is the class, i=1,2,3
- Try to classify a sample considering the probabilities of each classifier:
  - P(y=🔴| x; w)
  - P(y=🟩| x; w)
  - P(y=▲| x; w)
- To make a prediction we have to pick the class i that maximize $\boldsymbol{h_w^{(i)}(x)}$

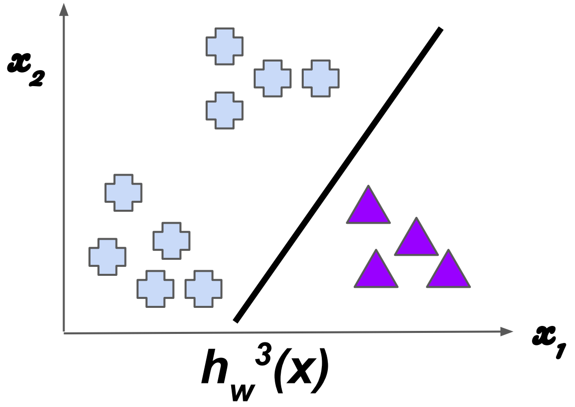
Multi-class classification: One-vs-ALL



Dataset







How to evaluate a classification task?

- Evaluating a classifier is often significantly tricker than evaluating a regressor
- Several metrics are possible but their use depends on the case:
  - Confusion matrix
  - Accuracy
  - Precision
  - Recall
  - F-1 measure
  - ROC Curve
- So…grab a coffee before

BREAK

Example: Iris plants dataset

Number of Instances

150 (50 in each of three classes)

Number of Attributes

4 numeric, predictive features and the class

Features Information

- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm
- **class:**
  - Iris-Setosa
  - Iris-Versicolor
  - Iris-Virginica

Iris Versicolor Iris Setosa Iris Virginica

Confusion matrix

- The diagonal elements represent the number of points for which the predicted label is equal to the true label. The off-diagonal elements are those that are mislabeled by the classifier.
- Perfect classification: diagonal matrix

Total		Predicted		
		Iris-setosa	Iris-versicolor	Iris-virginica
Real class	Iris setosa	14	1	1
	Iris-versicolor	0	16	0
	Iris-virginica	0	1	15

Accuracy = (14+16+15)/48 = 0,9375

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_true, y_pred)
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_true, y_pred)
```

Accuracy

- To evaluate a classification model we can use the **accuracy**
- The accuracy is defined as the ratio among the number of correct predictions over the total number of predictions
- The accuracy is often expressed as a percentage
- If the test-set is unbalanced we must use other metrics

Accuracy: Balanced test-set required

- Suppose you want to classify if in an image there is a cat or a dog

- Your dataset has 800 images of cats and 200 images of dogs

- We use 100 images for the test, but in this test-set we randomly selected:
  - 90 cats
  - 10 dogs

- The accuracy on the test-set is 90%
  - What does this accuracy mean?
  - If I use a “dummy” script that says every time “cat” without any learning it reaches the same accuracy
  - But if the test-set were balanced the “dummy” script would reach only 50% of accuracy

Accuracy: Balanced test-set required

- If I use a “dummy” script that says every time “cat” without any learning it reaches the same accuracy
  - But if the test-set were balanced the “dummy” script would reach only 50% of accuracy
  - **That’s why the test-set MUST BE BALANCED if you want to measure the accuracy!**

- What about the training-set ? Do we have to balance also it ?
  - It depends, if the training-set is unbalanced probably the less representative class will be less learnt by the model
  - It means that the model will probably predict that class less than the others because it will be less confident with that class
  - Usually it is better to have an almost balanced training-set, but sometimes we could desire to predict a label less than another
  - So finally, the answer it remains: It depends…

If unbalanced test (validation)

- We need some metrics that does not take into account only how many times the classifier gives the correct prediction in general but a metric we can compute for each class

- Let M be the chosen metric
- Once we measured M for each class (so the we do not suffer the unbalanced condition) we can resume a final measure for the classifier as an average among the ones we got for each class. In some cases we can report only M for each class without a final measure for the entire classifier

- The most simple case is the binary classification case where we can compute M for the “positive” label or for both “positive” and then for the “negative”

Precision

- In a binary classification we can divide predictions in True Positive (TP), True Negative (TN), False positive (FP), False Negative (FN)

		Predicted	
		Negative	Positive
Real class	Negative	TN	FP
	Positive	FN	TP

$$\text{Precision} = \frac{TP}{TP+FP}$$

- It is like an accuracy on a single class
- Alone does not give enough information since a trivial way to have a perfect (precision = 1/1 = 100%) is to use a test-set with just one example
- So usually precision is used along with another metric called Recall that takes into account the False negative

```
from sklearn.metrics import precision_score
precision_score(y_true, y_pred)
```

Recall and F-1 Score

- In a binary classification we can divide predictions in True Positive (TP), True Negative (TN), False positive (FP), False Negative (FN)

		Predicted	
		Negative	Positive
Real class	Negative	TN	FP
	Positive	FN	TP

$$\text{Precision} = \frac{TP}{TP+FP}$$

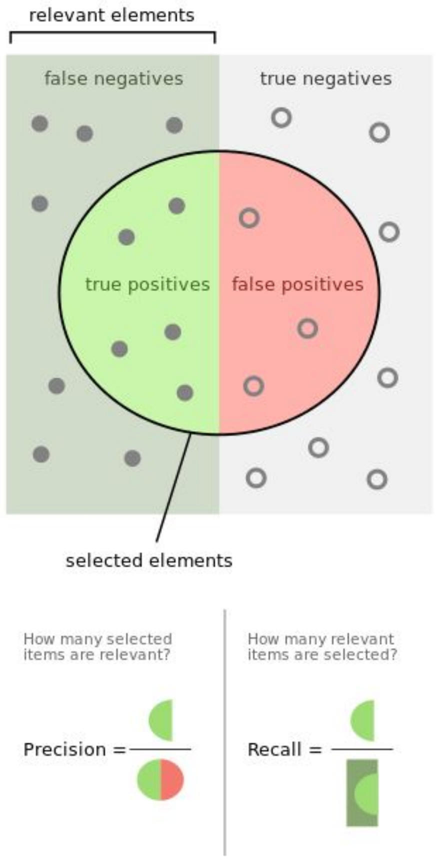
$$\text{Recall} = \frac{TP}{TP+FN}$$

$$F_1 = \frac{2}{\frac{1}{Precision}+\frac{1}{Recall}}$$

```
from sklearn.metrics import recall_score
recall_score(y_true, y_pred)
```

```
from sklearn.metrics import f1_score
f1_score(y_true, y_pred)
```

To summarize



- Precision can be thought as the accuracy of the positive predictions
- Recall (also known as Sensitivity or True positive rate) is the ratio of positive instances that are correctly detected by the classifier
- F1 is the harmonic mean of precision and recall. Whereas the regular mean treats all values equally, the harmonic one gives much weight to low values. As a consequence we can get a high F1 score only if both precision and recall are high

For the other metrics: see sklearn documentation: https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics

How to choose among the metrics?

- It depends (again)
- F-1 is a good compromise because it favors classifiers that have similar precision and recall. However, sometimes we would prefer to have a different value of precision and recall

- Let’s make some examples (from A.Gèron):

- You trained a classifier to detect videos that are safe for kids
    - You would probably prefer a classifier that rejects many good videos (low recall) but keeps only safe ones (high precision) rather than a classifier that has a much higher recall but lets a few really bad videos show up in your product

- You trained a classifier to detect shoplifters on video-surveillance images
    - It is probably fine if your classifier has only 30% precision as long as it has 99% recall
        - Sure, the security guards will get a few false alerts, but almost all shoplifters will get caught!

Example of Logistic Regression

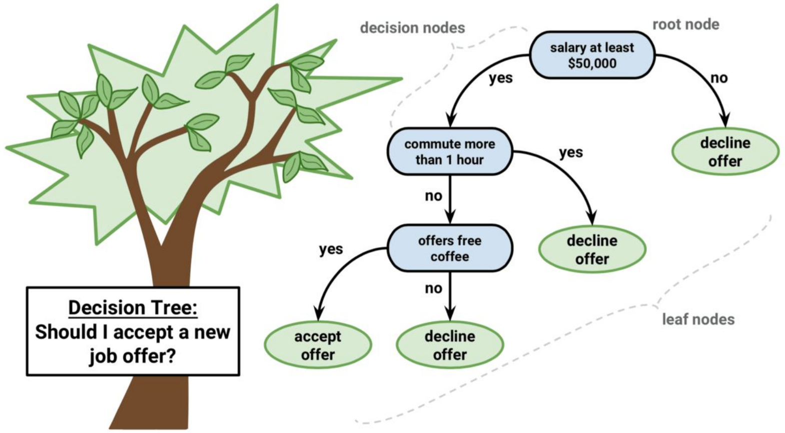
```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

# import some data to play with
iris = datasets.load_iris()
X = iris.data  # we only take the first two features.
y = iris.target

logreg = LogisticRegression()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
```

Iris plants

```
# Create an instance of Logistic Regression Classifier and fit the data.
logreg.fit(X_train, y_train)

# Make predictions using the testing set
y_pred = logreg.predict(X_test)
acc= accuracy_score(y_test, y_pred)
print(acc)
```

What is a decision tree ?

- Machine learning model with a flowchart-like structure
- Each internal nodes represents a “test” on a feature of our dataset
- Final nodes (leaf) are the predicted class or value
- Available both for classification and regression tasks



Example: Iris dataset

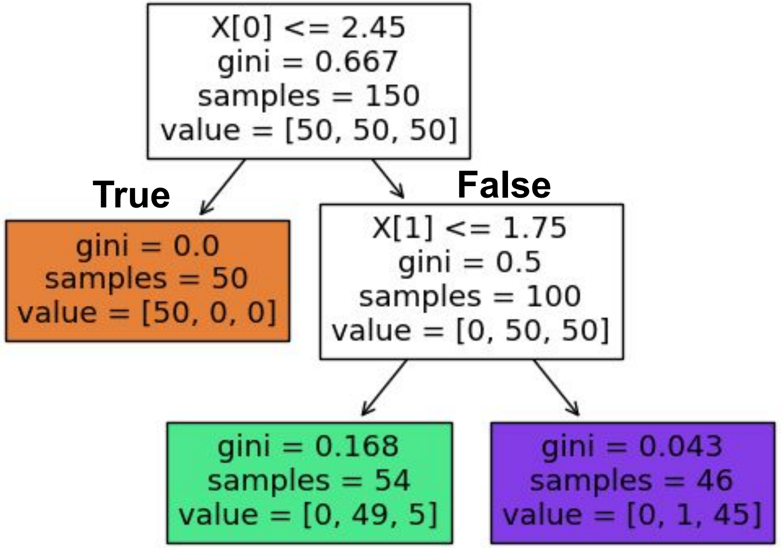
```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier,plot_tree
import matplotlib.pyplot as plt

iris=load_iris()
X =iris.data[:,2:] #petal length and width
y = iris.target

clf=DecisionTreeClassifier(max_depth=2)
clf.fit(X,y)
plot_tree(clf, filled=True)
plt.show()
```

Iris decision tree

- X[0] : petal length
- X[1]: petal width
- Gini: Impurity metric
  - A node is “pure” if Gini=0
    - If all training instances it applies to belong to the same class

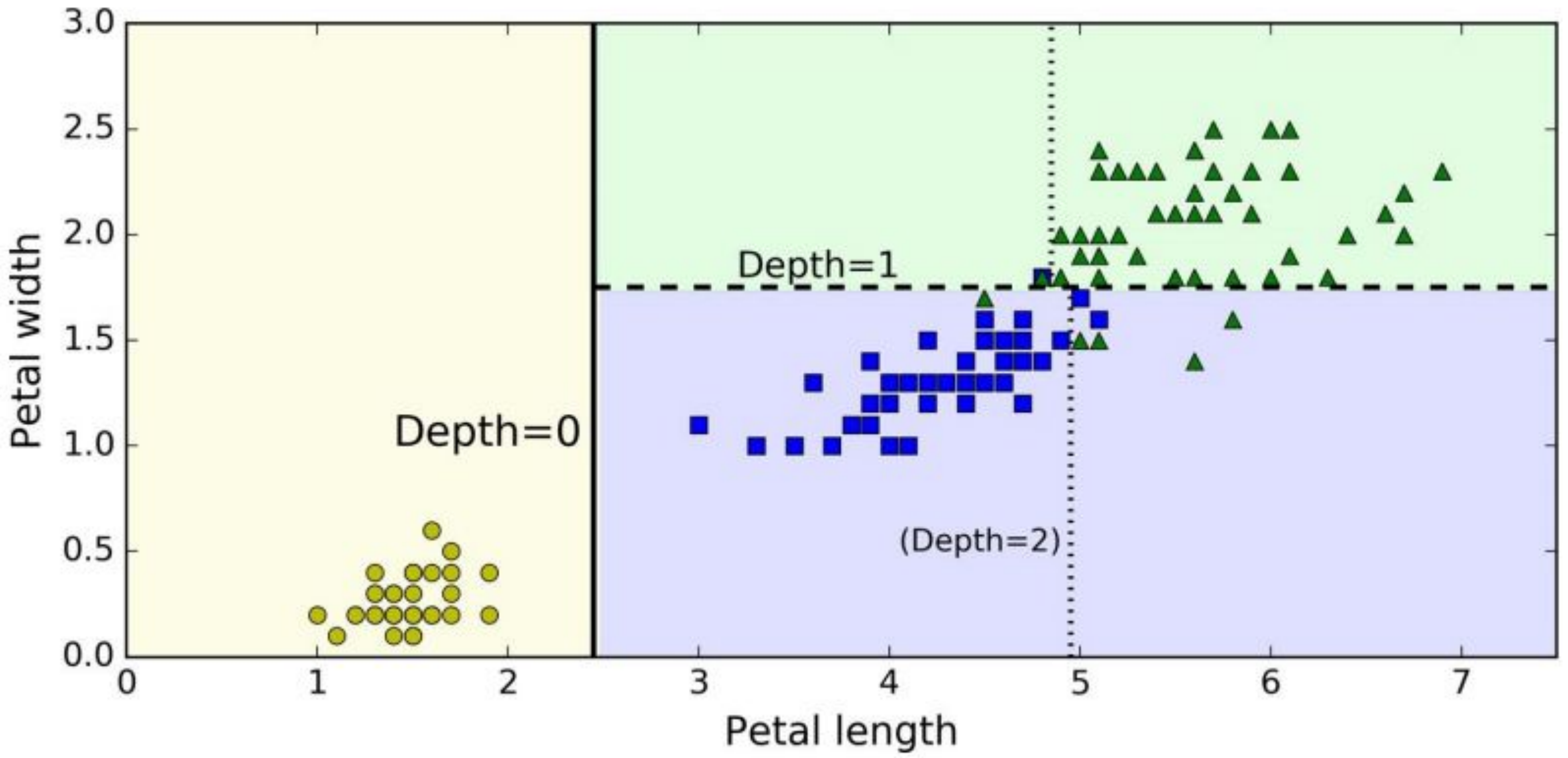
$$G_i = 1 - \sum_{k=1}^{n} {p_{i,k}}^2$$

- $p_{i,k}$ is the ratio of class k instances among the training instances in the i^{th} node.



- Gini for the depth-2 left node is $1 - (0/54)^2 - (49/54)^2 - (5/54)^2 = 0.168$
- Ratios corresponds also to output probabilities:
  - 0% for Iris-setosa
  - 90.7% for iris-Versicolor
  - 9,3% for iris-Virginica

Decision boundaries



CART training algorithm

- CART (Classification and Regression Tree algorithm)
- Simple idea:
  a. Splits the training set in two subsets using a single feature k and a threshold t_k (e.g., “petal length <= 2.45”)
  b. To choose k and t_k it looks for the purest subsets (weighted by their size) minimizing a cost function
  c. It stops once it reaches the maximum depth selected as a parameter

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

$$\text{where } \begin{cases} G_{\text{left/right}} \text{ measures the impurity of the left/right subset,} \\ m_{\text{left/right}} \text{ is the number of instances in the left/right subset} \end{cases}$$

Gini VS Entropy

- Another impurity measure is the Entropy
  - In Thermodynamics is a measure of molecular disorder and it is zero when molecules are well ordered
  - In Information Theory, it measures the average information content of a message (Shannon’s theorems) and it is zero when all messages are identical
- In Machine Learning a set’s entropy is zero when it contains instances of only one class.

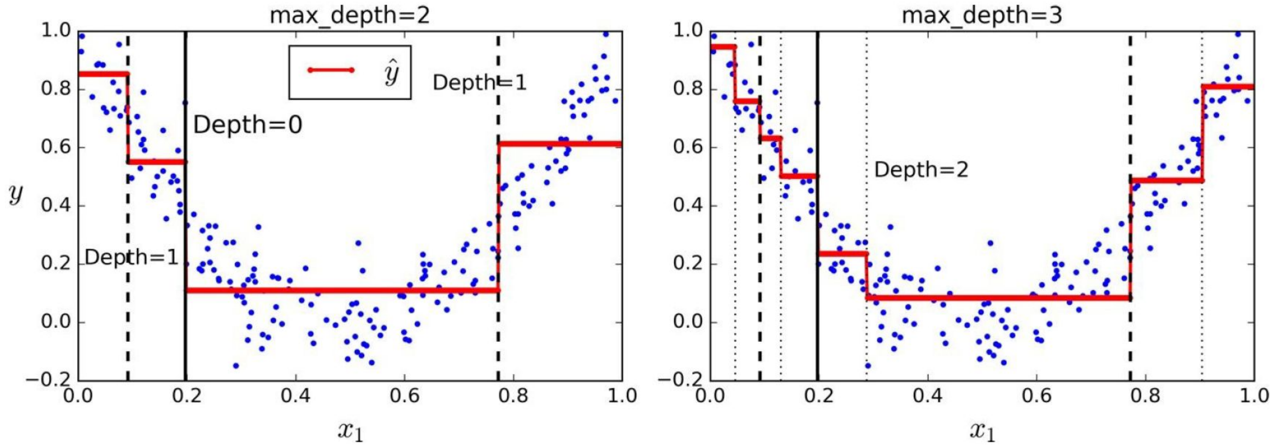
$$H_i = -\sum_{\substack{k=1 \\ p_{i,k} \neq 0}}^{n} p_{i,k} \log \left(p_{i,k}\right)$$

- Most of the time choosing Gini or Entropy does not introduce a big difference. Giny is usually faster to compute and tends to isolate the most frequent class in its own branch of the tree, while Entropy tends to produce slightly more balanced trees

Regression with Decision trees

- The algorithm splits each region in a way that makes most training instances as close as possible to the value to be predicted

- In sklearn import DecisionTreeRegressor



Predictions of two Decision Tree regression models

Regression with Decision trees

- The CART algorithm works exactly the same way as earlier, except that instead of trying to split the training set to minimize the impurity, it now tries to split to minimize the Mean Squared Error (MSE)

$$J(k, t_k) = \frac{m_{\text{left}}}{m}\text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m}\text{MSE}_{\text{right}} \quad \text{where} \begin{cases} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} \left(\hat{y}_{\text{node}} - y^{(i)}\right)^2 \\ \hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$$

Problems: Overfitting

- Decision trees make less assumptions about the training data with respect to linear models

- If left unconstrained the tree structure will adapt itself to the training-set very well !
  - Actually, most of the time it overfits the training-set !

- Decision trees are often called *nonparametric models*
  - Not because they don't have any parameters but because the n° of parameters is not determined prior to training
  - The model structure is free to stick closely to the data

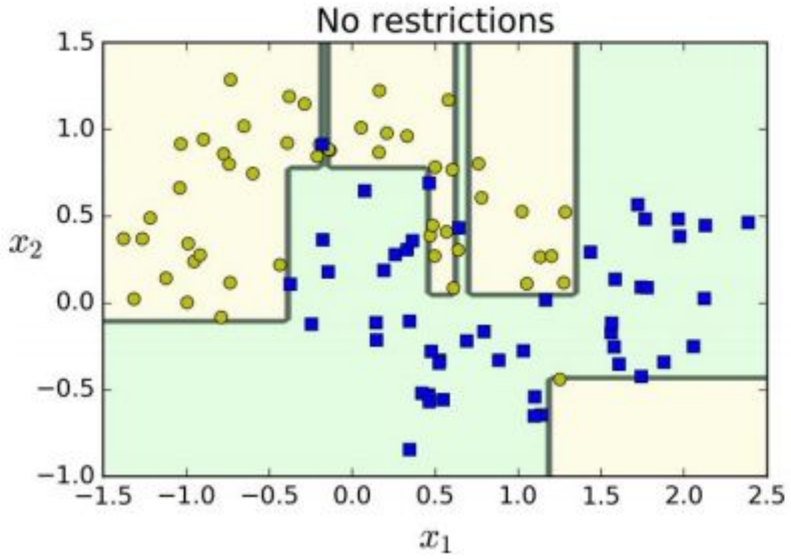
- On the opposite, a parametric model (such as linear model) has a predetermined number of parameters, it has a limited degree of freedom and that is useful to reduce the overfitting (but increasing the risk of underfitting)

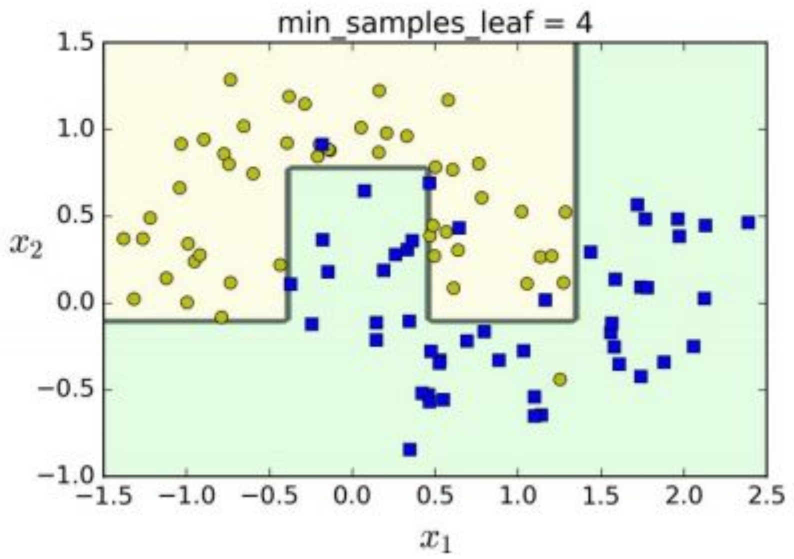
Solution: Regularization

- One solution can be reduce the degree of freedom of the model during the training phase (Regularization)
  - Select a max depth
  - Select a a minimum number of samples a leaf node must have
  - Select a max feature number that are evaluated for splitting each node

  - Pruning: First training the tree without restrictions and then deleting unnecessary nodes.
    - For example, a node whose children are all leaf nodes can be considered unnecessary if the purity improvement it provides it si not statistically significant (the improvement is purely random and it’s measured with a p-value

Regularization with min sample leaf





Regularization in Sklearn

- The DecisionTreeClassifier offers the following parameters:
  - min_samples_split: Select a a minimum number of samples a leaf node must have before it can be split
  - min_samples_leaf: Minimum number of samples a leaf node must have
  - min_weight_fraction_leaf: Same as min_samples_leaf but expressed as a fraction of the total number of weighted instances
  - max_leaf_nodes: Maximum number of leaf nodes
  - max_features: Maximum number of features that are evaluated for splitting each node