Biomedical Engineering Degree

4. Non-parametric Methods

Felipe Alonso Atienza
✉felipe.alonso@urjc.es

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universidad Rey Juan Carlos

References

1. J. Gareth, D. Witten, T. Hastie, and T. Tibshirani. *An Introduction to Statistical Learning*. Springer Texts in Statistics.
   - Chapter 4.6
   - Chapter 8
2. T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics.
   - Chapter 10
3. *scikit-learn documentation*

Outline

Parametric models

- They have a **number of parameters** ($\boldsymbol{\omega}$, in this case) that need to be learned
  - Linear regression:

$$\hat{y} = f_{\boldsymbol{\omega}}(\mathbf{x}) = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \dots$$

  - Logistic regression:

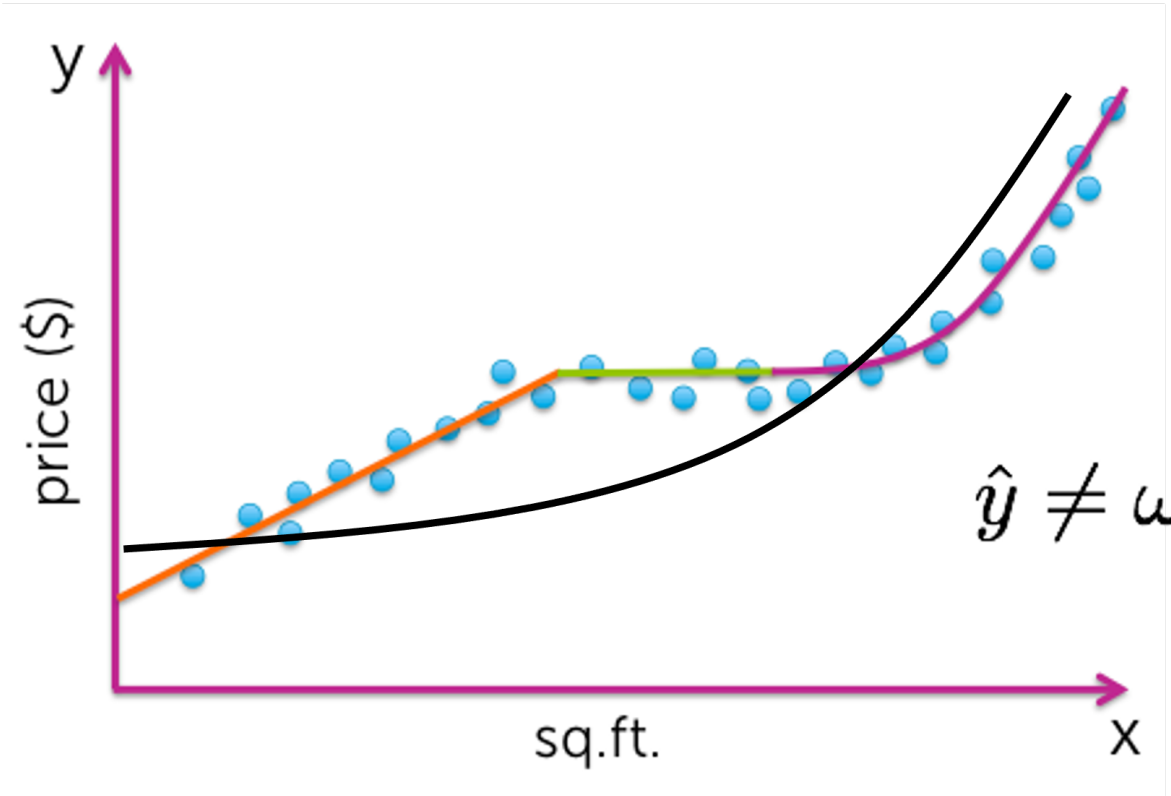
$$P(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-\boldsymbol{\omega}^T \mathbf{x}}} = \sigma(\boldsymbol{\omega}^T \mathbf{x})$$

☺ Easy to fit, simple interpretation

☹ However, it makes assumptions about how data is distributed

Motivating example

- Parametric regression: parametric models might not be flexible enough



$$\hat{y} \neq \omega_0 + \omega_1 x + \omega_2 x^2$$

Outline

$K-$Nearest neighbors ($k-$NN)

- One of the simplest methods of machine learning.
- It can be used for both **classification** (binary or multiclass) and **regression**.
- Intuitively, $k-$NN finds a predefined number (k) of training samples **closest** to a new point, and predict the label/value from these
  - Thus, $k-$NN assumes that similar things exist in close proximity. In other words, similar things are near to each other

CLASSIFICATION

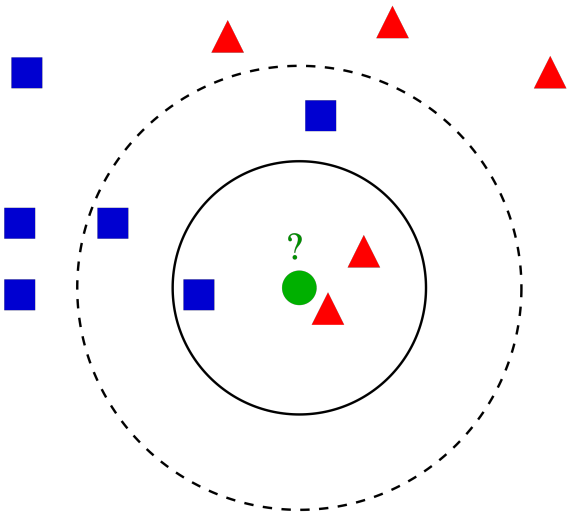


Image extracted from wikipedia

REGRESSION

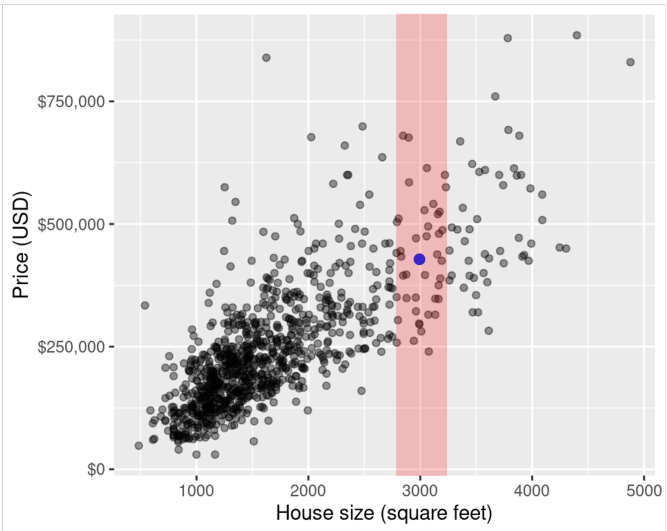


Image extracted from here

$K-$Nearest neighbors

To design the model, it is required:

- The value of k
  - it has to be set *a priori* and does not need to be learnt: **hyperparameter**
- A **distance measurement** that allows for the measurement of "similarity/likeness" between samples. Commonly, the **Euclidean** distance is used

$$d(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{\sum_{i=1}^{D} \left((x_1)_j - (x_2)_j\right)^2} = ||\mathbf{x}_1 - \mathbf{x}_2||_2$$

- The training set (fit method)

class sklearn.neighbors.KNeighborsClassifier(*n_neighbors=5*, ***, *weights='uniform'*, *algorithm='auto'*, *leaf_size=30*, *p=2*, *metric='minkowski'*, *metric_params=None*, *n_jobs=None*) [source]

class sklearn.neighbors.KNeighborsRegressor(*n_neighbors=5*, ***, *weights='uniform'*, *algorithm='auto'*, *leaf_size=30*, *p=2*, *metric='minkowski'*, *metric_params=None*, *n_jobs=None*) [source]

$K-$Nearest neighbors

To predict new data:

- The distance to each training case is calculated
- The k nearest neighbors are identified
- The output is calculated ...
  - ... as the majority class among its neighbors (**voting**)
    - `weights` can be used to solve ties.
  - ... as the **average** of the values of its neighbors (MSE)
    - ... as the **weighted average** of the values of its neighbors
    - ... as the **median** of the values of its neighbors (MAE)
    - ... as the **mode** of the values of its neighbors (MAE)

$K-$Nearest neighbors

Example

We have a set of 5 two-dimensional observations (x_1, x_2), for which we know the desired output y, as shown in the following table:

x_1	x_2	y
1	0.3	4
1.5	0.5	6
3.1	-0.1	6
5	-0.2	3
6.2	0.4	2
4	0	$-$

From these 5 observations we wish to design a regressor using a $k-$NN algorithm with Euclidean distance, in order to estimate the output for the observation of coordinates $(x_1 = 4, x_2 = 0)$. If $k = 3$, what would be the estimated output?

Exercise

Could you write a python code to solve this example?[a]

[a] This blog might help: well coded, with a link to recommender systems

$K-$Nearest neighbors: choosing k

Example

Given the values of $k = 1, 10, 100$, which value of k corresponds to each figure? Can you explain why?

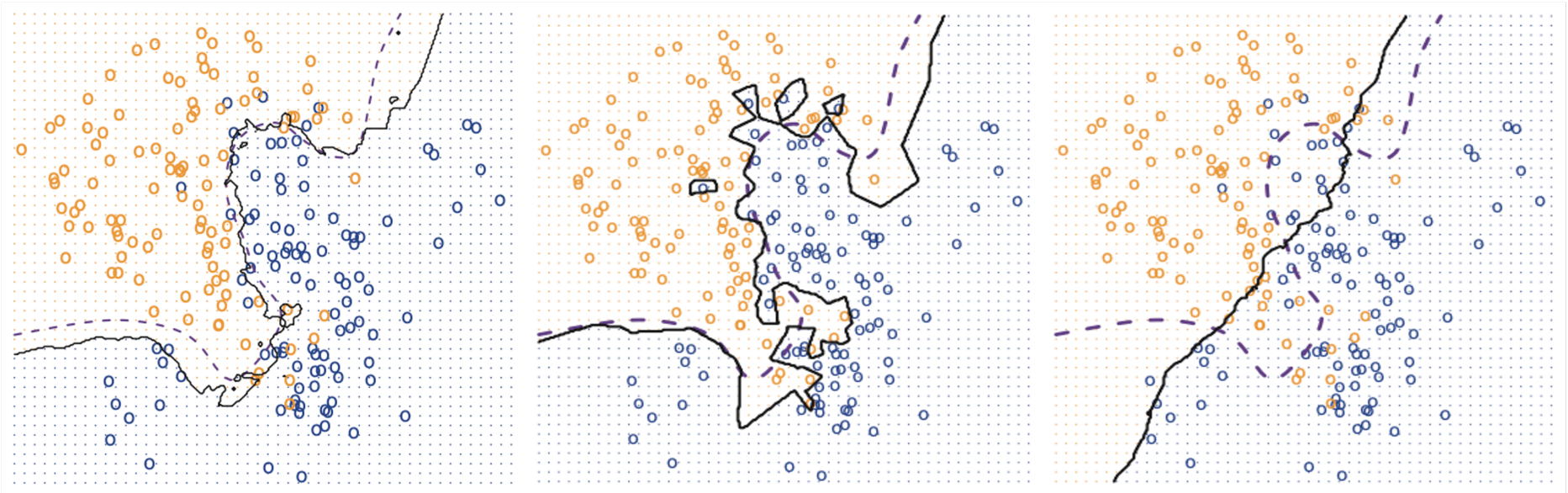


Image extracted from An Introduction to Statistical Learning

$K-$Nearest neighbors: choosing k

- So, how do we select k? **Cross-validation**

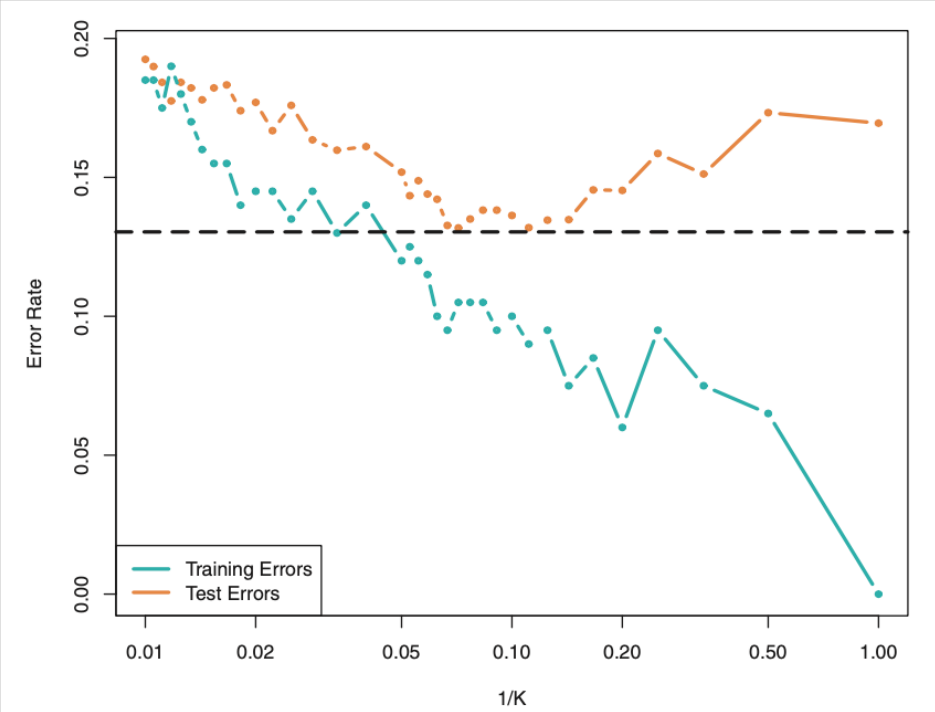


Image extracted from An Introduction to Statistical Learning

- **Small values** of k will produce highly complex boundaries and models that will tend to overfit
- **Large values** of k may result in too soft a boundary that also does not generalize properly

$K-$Nearest neighbors: choosing k

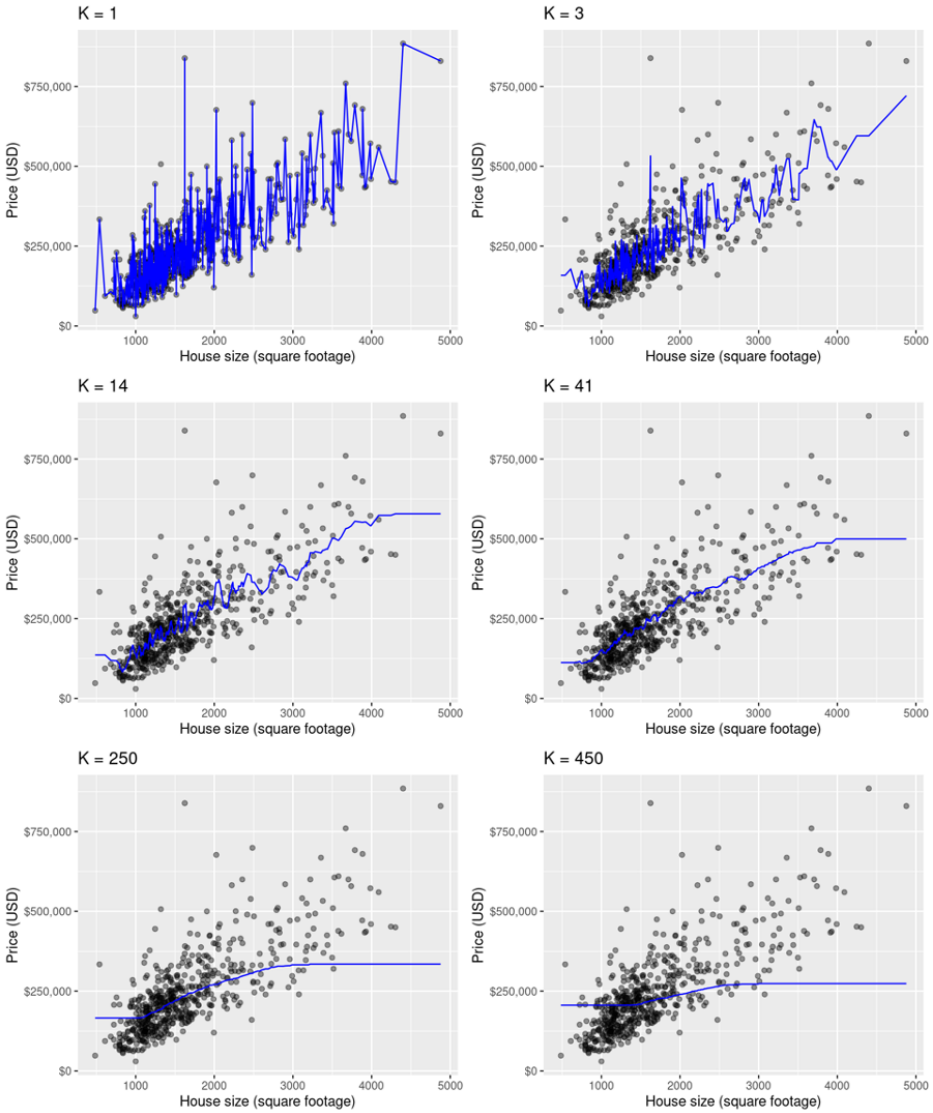


Image extracted from here

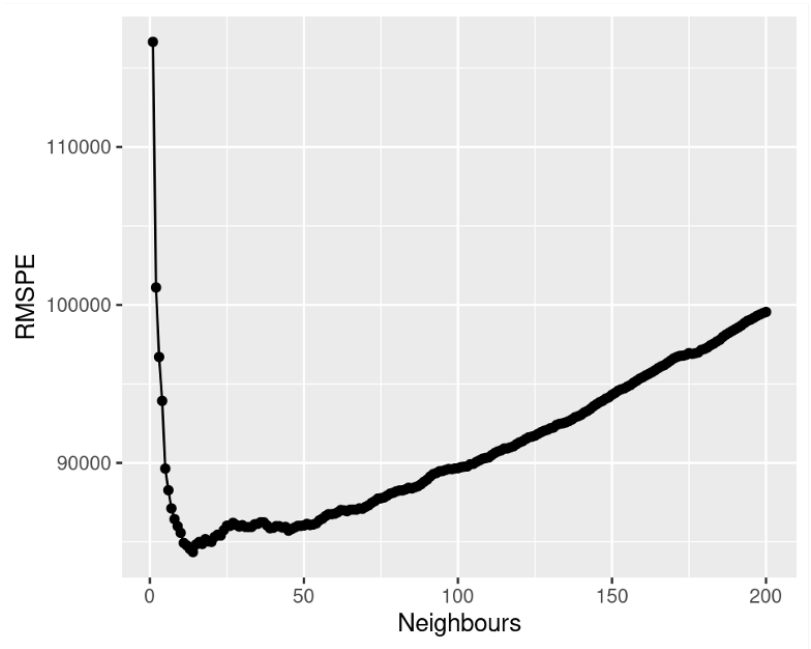


Image extracted from here

$K-$Nearest neighbors: pros and cons

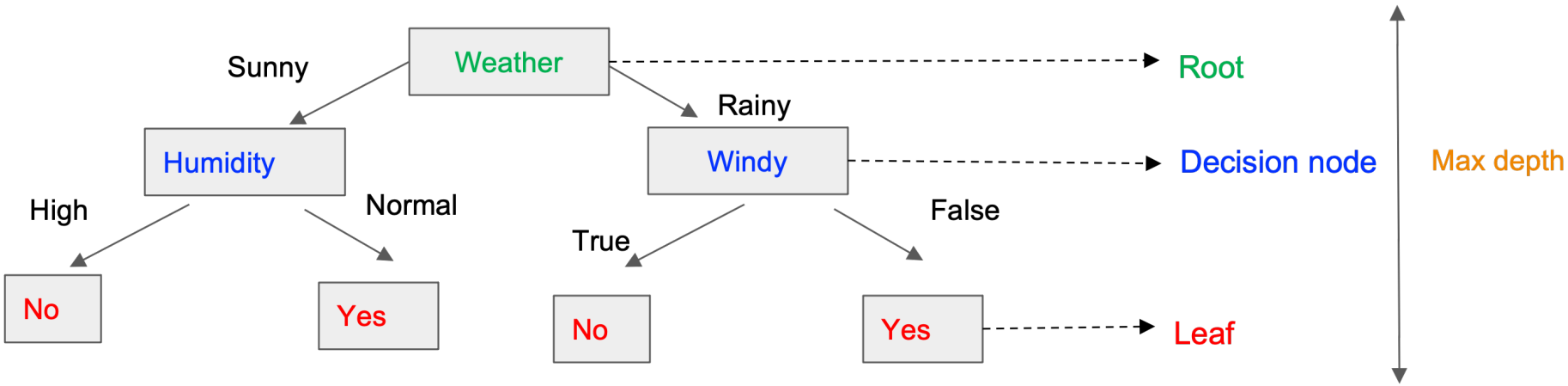
- ☺ Simple and easy to understand
- ☺ No assumptions about how data must look like
- ☺ Works well with non-linear relationships
- ☺ It can be extended to multiple predictors

- ☹ **Variables need to be scaled/standarized**: distance-based algorithm
- ☹ Poor performance when data is sparse (few observations) and large number of features (high dimensionality): **curse of dimensionality**
- ☹ Boundary problems: it does not predict well beyond the range of values input in your training data
  - Weight points by the inverse of their distance
- ☹ Slow when high number of observations: distance among instances must be calculated

Outline

Decision Trees (DTs)

- A non-parametric supervised learning method used for **classification** and **regression**.
- The objective is to create a model that predicts the value of a target variable by learning **simple decision rules**[1] inferred from the data features
  - Example: will you play tennis tomorrow?



[1] These are also called *decision stumps*

Decision Trees (DTs): regression

- Motivating example: predict a baseball player's log salary[2] based on:
  - `Years`: #years that he has played in the major leagues
  - `Hits`: #hits that he made in the previous year

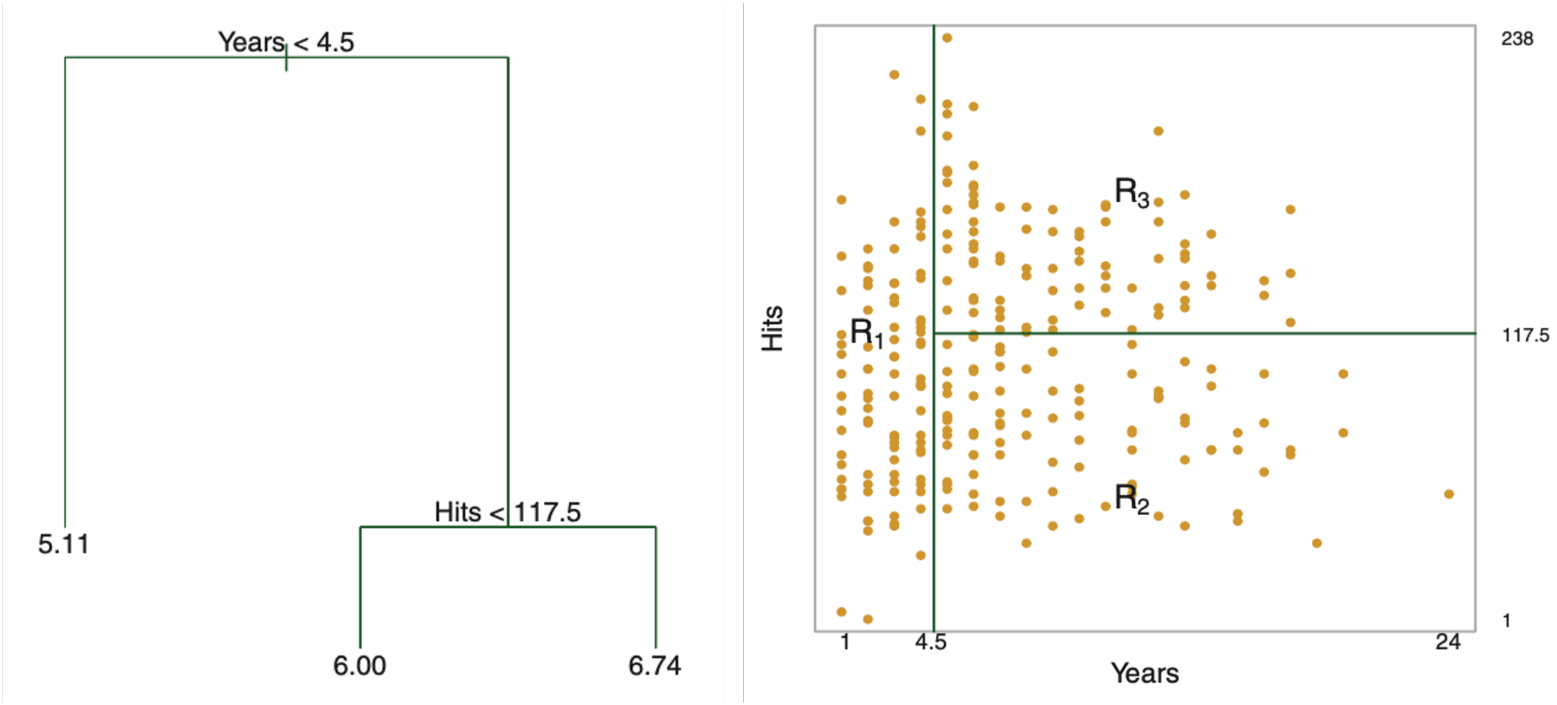


Image extracted from An Introduction to Statistical Learning

- `Years` can be interpreted as the **most important** factor

[2] Related examples appears in the movie moneyball, and the book the signal and the noise

How a DT is built?[3]

Given a training set $\{\mathbf{X}, \mathbf{y}\}$, with $\mathbf{X} \in \mathbb{R}^{\mathbf{N}\times\mathbf{D}}$ and $\mathbf{y} \in \mathbb{R}^{\mathbf{N}\times\mathbf{1}}$

1. Start with the root node, that contains all training samples
2. Apply a **threshold test** on the root node, so **two branches** are created
   - Select the predictor $\mathbf{x_d}$ and the threshold s that **best** splits the predictor space into two regions
3. Recursive splitting: on each branch, apply a threshold test
   - if some **stopping criterion** is fulfilled, a leaf node is created.
   - Otherwise, continue growing the tree on this branch (go to step 2)

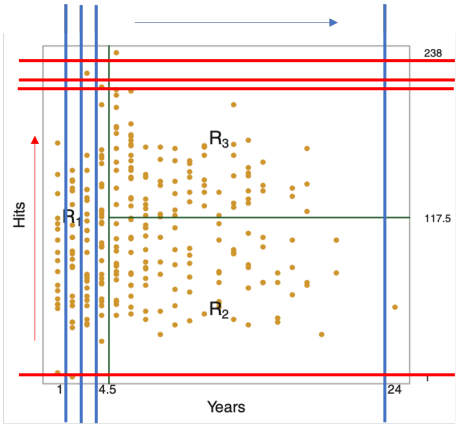
Note that there are different algorithms to implement decision trees, some of them are: ID3, C4.5, or CART (Classification And Regression Tree). The latter is the one used by scikit-learn (binary decisions).

[3] Check this visualization

How a DT is built?

Threshold test: how to select a predictor $\mathbf{x_d}$ and a threshold s?

1. For each feature/predictor $d = 1, \ldots, D$ set all possible values of the threshold s



Original image extracted from An Introduction to Statistical Learning

   - For each threshold, evaluate a figure of merit:
     - ★ Regression: MSE
     - ★ Classification: Classification error, Gini index, Entropy gain

2. Select the predictor and threshold that provides the best figure of merit.

How a DT is built?

Classification metrics: let be a classification problem with K categories. Let define p_k as the proportion of training observations at a node n for the kth class.

- Classification error:

$$E = 1 - \max_k p_k$$

- **Gini index**, a measure of homogeneity (purity). A small value indicates that a node contains predominantly observations from a single class (purity)

$$G = \sum_{k=1}^{K} p_k(1 - p_k)$$

- **Cross-entropy**, also a measure of *purity*. It will take on a value near zero if the p_k's are all near zero or near one

$$D = -\sum_{k=1}^{K} p_k \log p_k$$

How a DT is built?

Stopping criteria: When should we stop partitioning?

- In theory, we could build the tree until every training samples is correctly predicted...
  - ... this increases the **complexity** of the tree (a deep tree)
  - ... which tends to **overfit** the data

- We need to control de size (complexity) of the tree (**prunning**)
  1. `max_depth`: the maximum depth of the tree
  2. `min_samples_leaf`: The minimum number of samples required to be at a leaf node
  3. ... and some **others**.

Will you increase or decrease `max_depth`, `min_samples_leaf` to control the size of the tree?

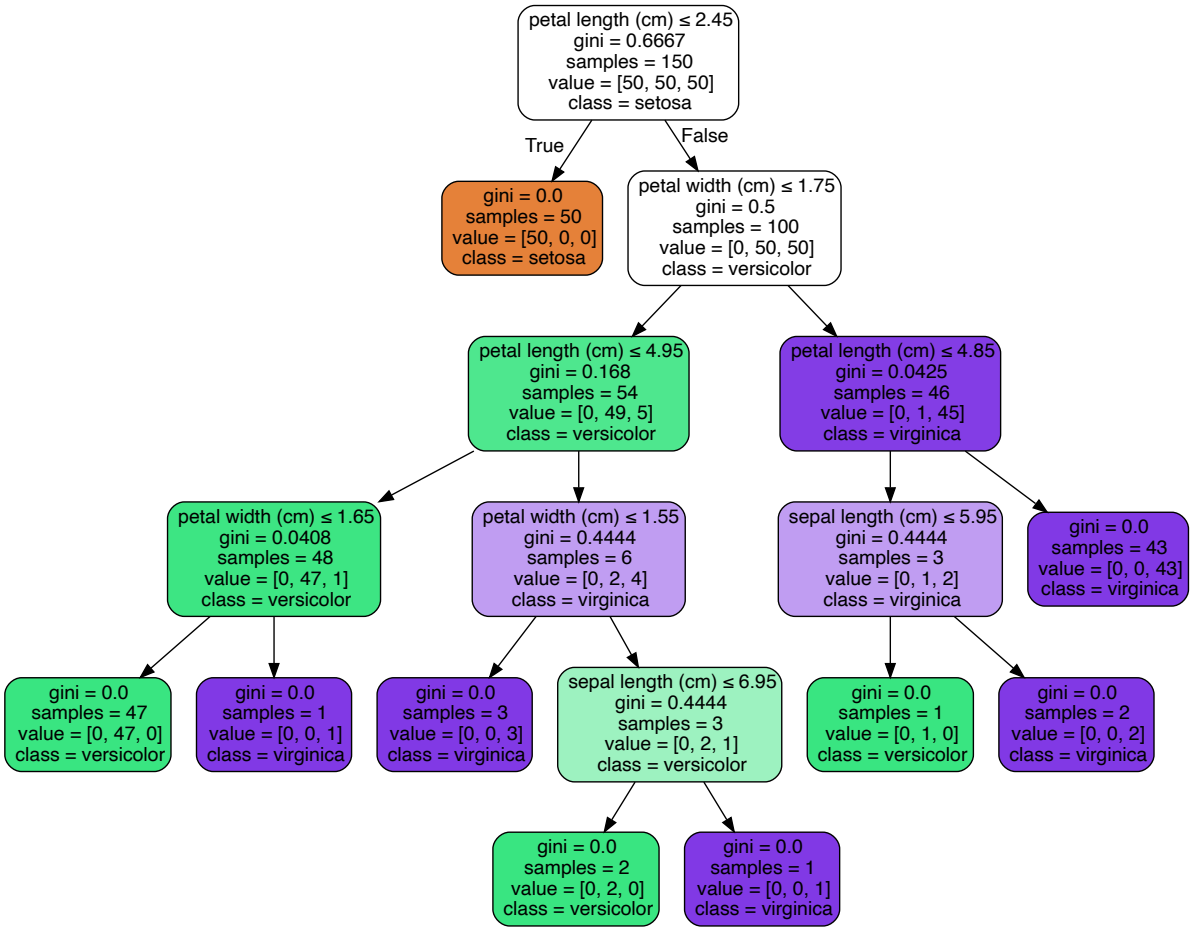
- Note that tree complexity parameters are hyper-parameters of the DTs, that need to be calculated using **cross-validation**

Decision path

- What would be the decision for a new observation with:

$$(\texttt{petal\_length} = 3\,\text{cm}, \texttt{sepal\_length} = 6\,\text{cm}, \texttt{petal\_width} = 1\,\text{cm})$$

- What would be the decision if the tree were $\texttt{max\_depth} = 3$



Original image extracted from An Introduction to Statistical Learning

DTs: pros and cons

- ☺ Simple to understand and to interpret. Trees can be visualized
- ☺ Variables do not need to be scaled/standarized
- ☺ Able to handle both numerical and categorical data (not in sklearn)
- ☺ It can be extended to multiple predictors
- ☺ Class probability can be estimated for each observation
- ☺ Feature importance

- ☹ They overfit easily, pruning methods are required
- ☹ They are not very robust. Small changes in training data can cause large changes in the trees built and their predictions.
- ☹ Prediction results can be worse than those obtained with other more advanced methods.

Outline

Ensemble methods

- Combine the predictions of several **base estimators** built with the **same training data** in order to improve generalizability / robustness over a single estimator by introducing diversity:
  - Use a given base estimator, and train it using different subsamples of the training set
  - Use different base estimators

- Combination can be done by
  - Taking the (weighted) mean of the predictions
  - Taking the mode of the predictions
  - Using the majority vote or the average predicted probabilities (soft vote)

- Statistically, we aim at reducing the variance of the estimator by averaging ...

What's the variance of the mean of N independent observations each with variance σ^2?

Ensemble methods

In the context of using different subsamples of the training set, there are two major families of ensemble methods

1. **Bagging** (Bootstrap[4] AGGregation): build several estimators independently and then **average** their predictions.
   - **Random Forest**: a specific bagging version built on decision trees.

2. **Boosting**: base estimators are **built sequentially** and one tries to reduce the bias of the combined estimator

[4] Bootstrap: sampling with replacement

Random Forest

We generate a forest of B decision trees whose predictions are combined to solve classification or regression tasks.

Random Forest Algorithm

for $b = 1, \ldots, B$:

1. Create a subsample with replacement of the training set
2. Train a DT on this subsample, with the following condition
   - For each split, a random sample of $m < D$ predictors is chosen as split candidates (each node only has access to a subset of the input predictors).

- Predictions are made using the forest (B trees):
  - Regression: averaging the individual B predictions
  - Classification: majority vote on the B predictions

Random Forest in scikit-learn

Besides the hyper-parameters that control the complexity of DTs, Random Forest introduces additional hyper-parameters to adjust:

- `n_estimators`: #trees in the forest (B)
  - The larger the better, but also the longer it will take to compute. Results will stop getting significantly better beyond a critical number of trees
  - Recommendation: use grid search on $B = 50, 100, 200, 500$ (might require fine tuning)
- `max_features`: size of the random subsets of features (m)
  - The lower the greater the reduction of variance, but also the greater the increase in bias.
  - Recommendation: use value by default ($m = \sqrt{D}$)
- There are other parameters to play with

Feature importance

- With the aggregation of trees, interpretability is lost.
- However, it is possible to extract a measure of the importance of each variable
  - How much the performance improves in the splits associated with that variable
  - In other words: for each split of each trained tree, the improvement in performance due to the variable by which the tree is partitioned.
- Relative measure: scaled between 0-100

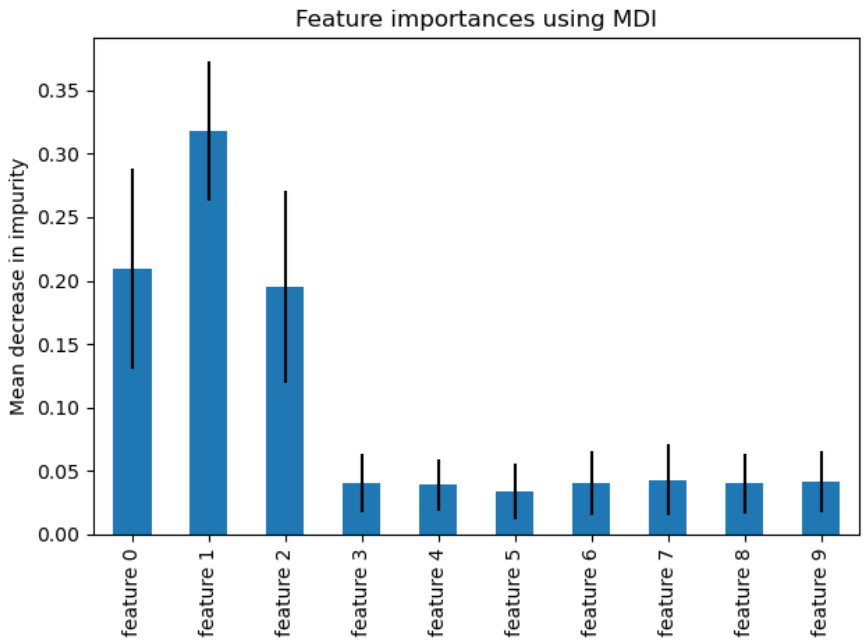


Image extracted from sklearn documentation

Boosting

- One of the most brilliant ideas in machine learning
- Iterative process based on linear combination of **weak learners**
- Applicable to any ML algorithm. Usually with DTs (**Boosted Trees**)
- One of the main winners in Kaggle competitions (if Deep Learning methods cannot be applied).

Boosting: basic idea

- Intuition:

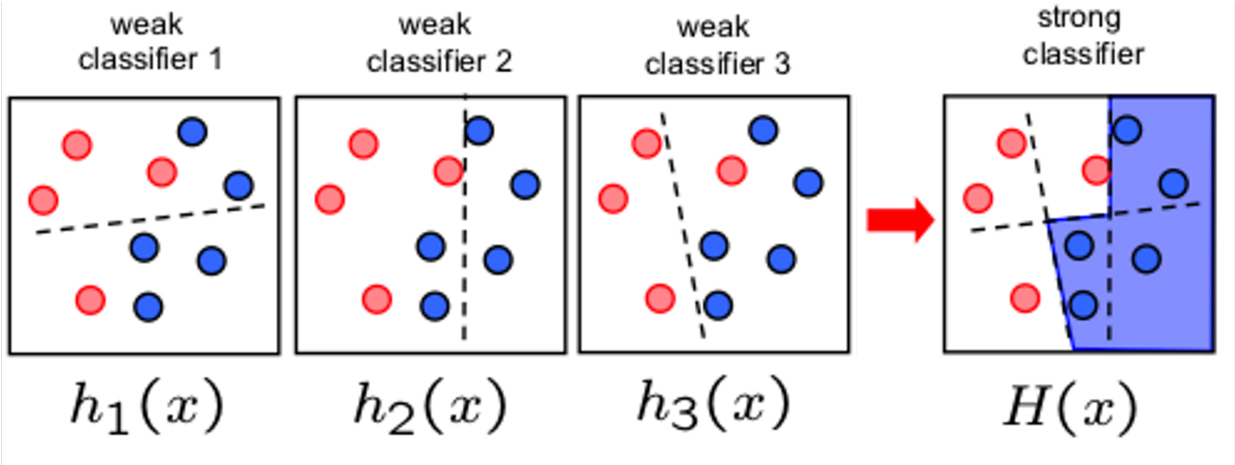


Image extracted from here

$$H(x) = \text{sign}\left(\alpha_1 h_1(\mathbf{x}) + \alpha_2 h_2(\mathbf{x}) + \alpha_3 h_3(\mathbf{x})\right)$$

- Iterative process

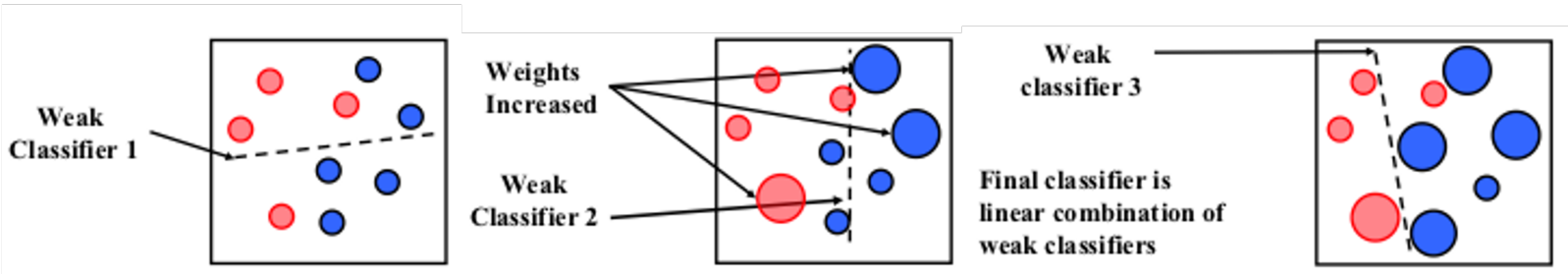


Image extracted from here

Boosted Trees: hyper-parameters

Using as reference sklearn implementation of Gradient Tree Boosting, which is a generalization of boosting to arbitrary differentiable loss functions

- `n_estimators`: #iterations
  - Large values tend to overfit
  - Recommendation: use grid search on $50, 100, 200, 500, 1000$ (might require fine tuning)

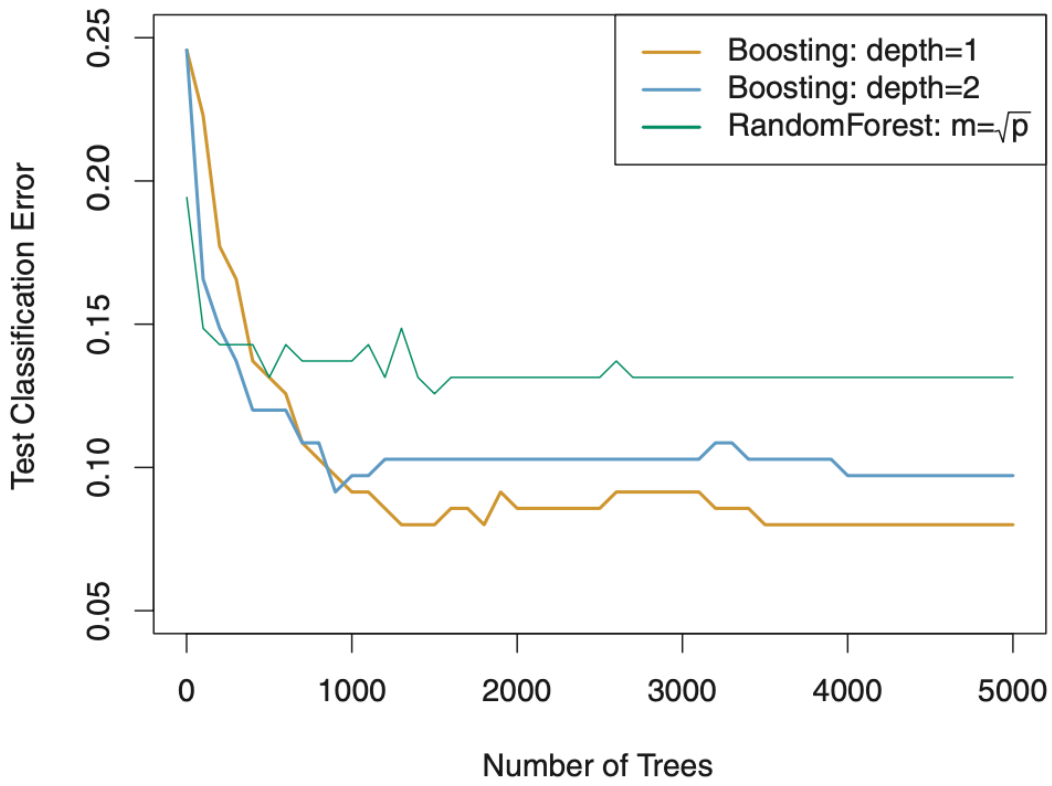
- `learning_rate`: shrinks the contribution of each tree. It is a positive number of small value, usually 0.01, 0.001
  - There is a trade-off between `learning_rate` and `n_estimators`. The smaller `learning_rate`, larger `n_estimators` is needed to converge[5]
  - Recommendation: use grid search on log-scale

- `max_depth`: to control tree complexity
  - Ideally a decision stump.
  - Recommendation: use grid search on small values

[5] When the performance metric stabilizes

Boosted Trees: hyper-parameters

Easy to overfit: "*power without control is useless*"



Original image extracted from An Introduction to Statistical Learning

Boosted Trees: other (well-known) implementations

- **XGBoost**
  - **Documentation**
  - **paper**
- **LightGBM**
  - **Documentation**
  - **paper**

Outline

Artificial Neural Networks

- Inspired by the network structure of biological neurons (**neural networks**)
- Building block of **state-of-the-art machine learning algorithms** in Artificial Intelligence areas such as
  - Computer vision
  - Natural Language Processing (NLP)
- **Performance higher than humans** in some applications
- When stacked in large architectures (deep networks), **they require quite some effort to train** ...

... OpenAI GPT-3 architecture ...

... has 175 billion parameters

... cost 11M dollars to trains

... produced the equivalent of 552 tons of CO2 during training

- They are indeed **parametric models**!

Biological neuron

- The biological neuron is composed of:
  - The cell body
  - The axon

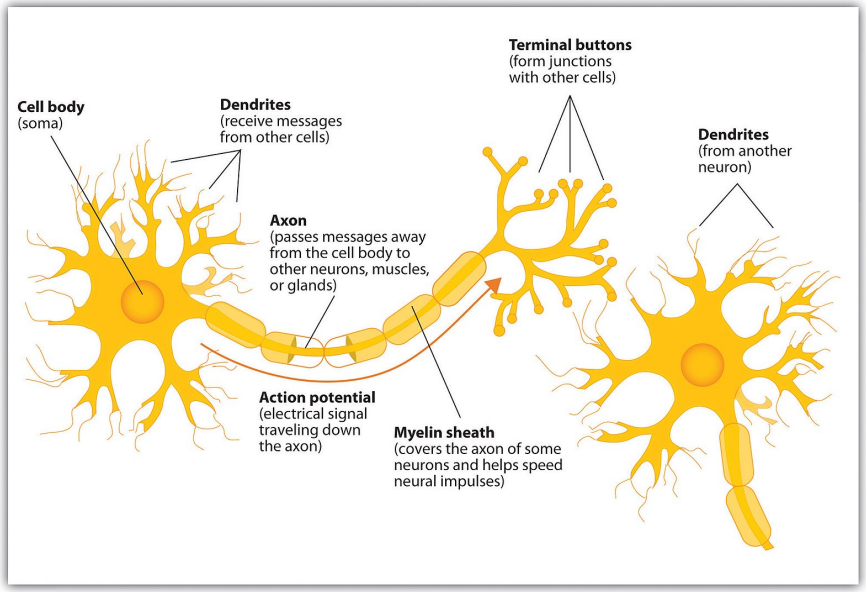


Image extracted from here

- Synapse: neurons receive electrical impulses (signals) from other neurons, and if these signals are strong enough they fire (active) surrounding neurons

Perceptron

- The first ANN architecture, invented by Frank Rosenblatt in 1958

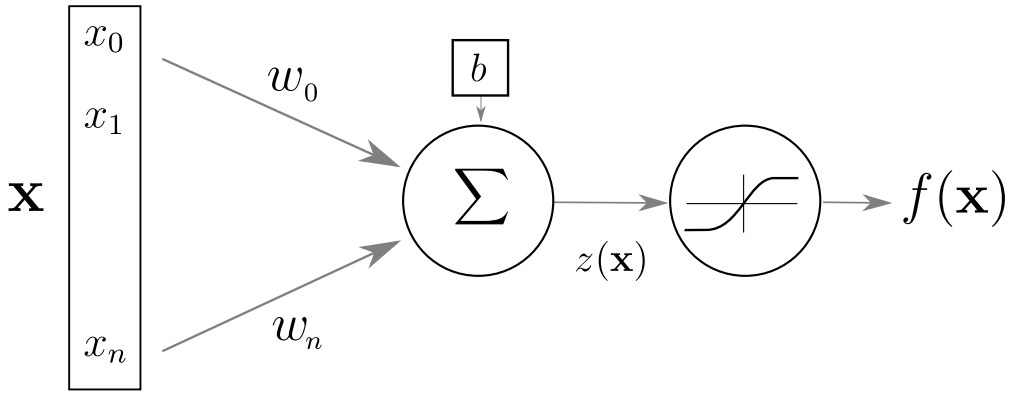


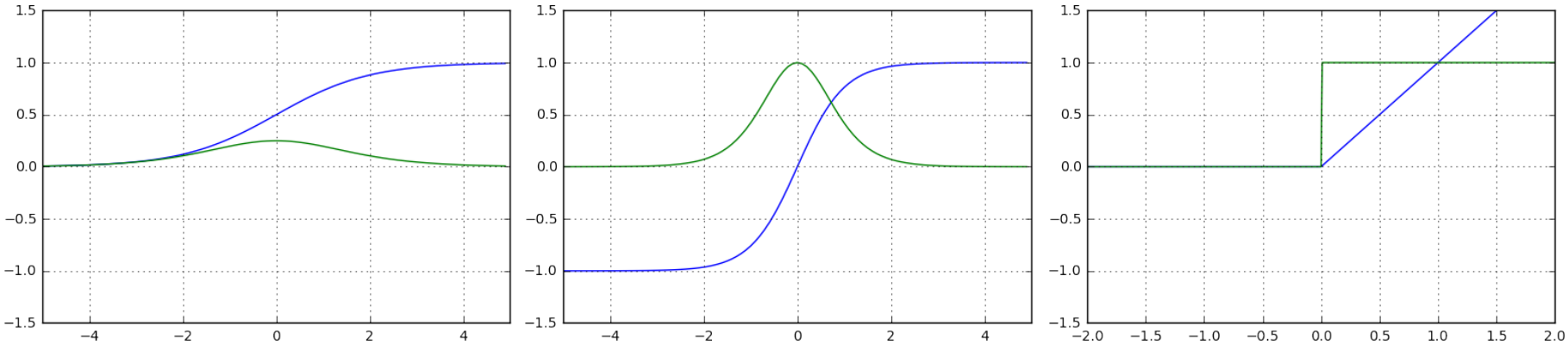
Image extracted from Deep Learning course

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

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

- It is a learning algorithm (binary linear classifier): update weights ($\mathbf{w}$) so that a cost function is optimized via Stochastic Gradient Descend

Activation functions



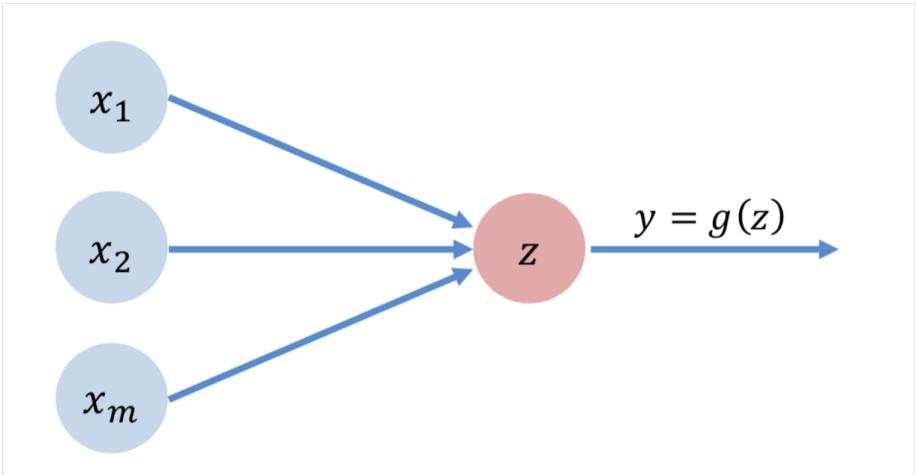
$$\text{sigm}(x) = \frac{1}{1+e^{-x}} \qquad \tanh(x) = \frac{e^{2x}-1}{e^{2x}+1} \qquad \text{relu}(x) = \max(0,x)$$

$$\text{sigm}'(x) = \text{sigm}(x)(1-\text{sigm}(x)) \qquad \tanh'(x) = 1-\tanh(x)^2 \qquad \text{relu}'(x) = 1_{x>0}$$

Image extracted from Deep Learning course

- Sigmoid and tanh functions are only really sensitive to changes around their mid-point of their input

Multilayer Perceptron (MLP)



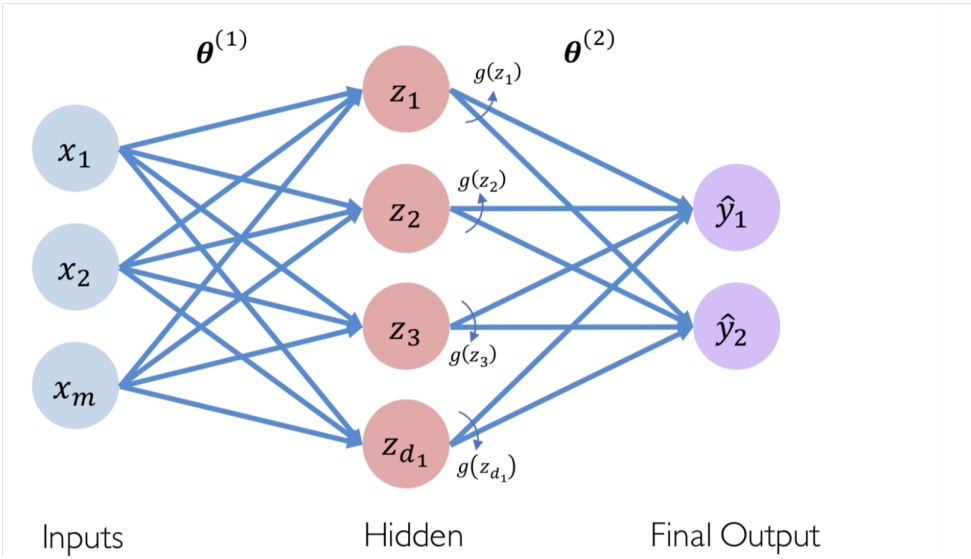


Image extracted from MIT 6.S191 Introduction to Deep Learning

- Perceptron has limited scope in complex tasks
- Combine perceptrons in multiple layers
- Input signal **forward propagates** to the output layer (each layer is feeding the next one)
- It is a universal approximator (any classification boundary)

How does the MLP learn?

Training: find $\mathbf{W}$ (includes bias term) that minimize the cost function

1. Start with random parameters $\mathbf{W}$
2. For E epochs perform:
   - Randomly select a small batch of K samples ($K < N$)
   - Compute the **loss function** L (forward propagation)
   - Compute gradients ($\nabla_{\mathbf{W}} L$) of the loss function[6] (backward propagation)
   - Update parameters (learning rate $\eta > 0$):

$$\mathbf{W} = \mathbf{W} - \eta \frac{1}{K} \sum_{k=1}^{K} \nabla_{\mathbf{W}} L_i$$

   - Measure train and validation accuracy/error
3. Continue until:
   - Loss function converge
   - Validation accuracy/error stops increasing/decreasing

[6] Gradients are calculated over all layers, and ReLU activation function prevents the vanishing gradient problem

Loss function in scikit-learn

- Classification (binary): Log-loss (as in logistic regression)

$$L(\mathbf{W}) = -\frac{\mathbf{1}}{\mathbf{N}} \sum_{\mathbf{n=1}}^{\mathbf{N}} \mathbf{y^{(i)}} \log\left(\mathbf{\hat{y}^{(i)}}\right) + (\mathbf{1} - \mathbf{y^{(i)}}) \log\left(\mathbf{1} - \mathbf{\hat{y}^{(i)}}\right)$$

  - Supports multi-class classification by applying Softmax as the output function
- Regression (no activation function in the output layer): MSE

$$L(\mathbf{W}) = \frac{\mathbf{1}}{\mathbf{N}} \sum_{\mathbf{n=1}}^{\mathbf{N}} \left(\mathbf{y^{(i)}} - \mathbf{\hat{y}^{(i)}}\right)^{\mathbf{2}}$$

Tips on practical use (scikit-learn)

- Scale/standarize your data: MLP is sensitive to feature scaling
- Use regularization
  - L2 regularization term in the cost function

$$L(\hat{y}, y, W) = -y \ln \hat{y} - (1 - y) \ln (1 - \hat{y}) + \alpha ||W||_2^2$$

$$L(\hat{y}, y, W) = \frac{1}{2} ||\hat{y} - y||_2^2 + \frac{\alpha}{2} ||W||_2^2$$

  - Early stopping
  - scikit-learn does not support dropout (deactivation of some random neurons during training)
- Use other optimization algorithm

class sklearn.neural_network.MLPClassifier(*hidden_layer_sizes=(100), activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000*) ¶ [source]

Image extracted from scikit-learn

Deep learning resources

- Frameworks: scikit-learn might not be the most suitable library for training deep neural networks
  - tensorflow
  - keras
  - pytorch
- Courses:
  - Deep Learning specialization
  - Fast.AI
  - Introduction to Deep Learning
  - ... and many others ...

Outline

Biomedical examples and applications

Check the research/publications area on ...

1. Google Health
2. Subtle Medical. AI-Powered Image Enhancement
3. Owkin Platform. Federated-learning
4. Oncora medical. Data to fight cancer
5. BioSymetrics. AI-Powered Drug Discovery

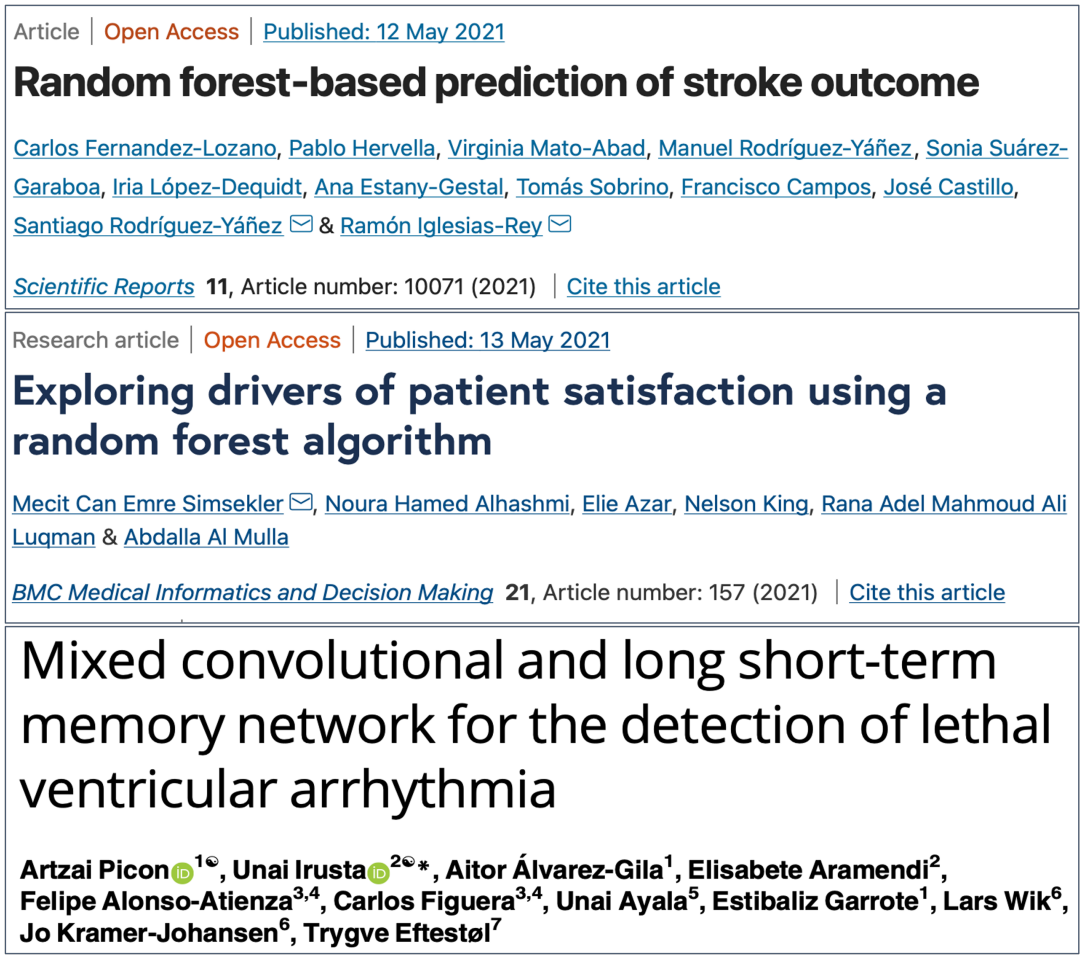
6. Curai Health. Affordable primary care. They have a tech blog.

7. Savana Medical. Clinical Natural Language Processing (NLP)
8. Idoven. Online home cardiology based on AI
9. Iomed. Fostering clinical research

10. Ted Talk: Emprendiendo para transformar la sanidad
11. Ted Talk: ¡Liberad los datos!

Biomedical examples and applications

... some papers ...



Article | Open Access | Published: 12 May 2021

Random forest-based prediction of stroke outcome

Carlos Fernandez-Lozano, Pablo Hervella, Virginia Mato-Abad, Manuel Rodríguez-Yáñez, Sonia Suárez-Garaboa, Iria López-Dequidt, Ana Estany-Gestal, Tomás Sobrino, Francisco Campos, José Castillo, Santiago Rodríguez-Yáñez & Ramón Iglesias-Rey

Scientific Reports **11**, Article number: 10071 (2021) | Cite this article

Research article | Open Access | Published: 13 May 2021

Exploring drivers of patient satisfaction using a random forest algorithm

Mecit Can Emre Simsekler, Noura Hamed Alhashmi, Elie Azar, Nelson King, Rana Adel Mahmoud Ali Luqman & Abdalla Al Mulla

BMC Medical Informatics and Decision Making **21**, Article number: 157 (2021) | Cite this article

Mixed convolutional and long short-term memory network for the detection of lethal ventricular arrhythmia

Artzai Picon[1], Unai Irusta[2]*, Aitor Álvarez-Gila[1], Elisabete Aramendi[2], Felipe Alonso-Atienza[3,4], Carlos Figuera[3,4], Unai Ayala[5], Estibaliz Garrote[1], Lars Wik[6], Jo Kramer-Johansen[6], Trygve Eftestøl[7]