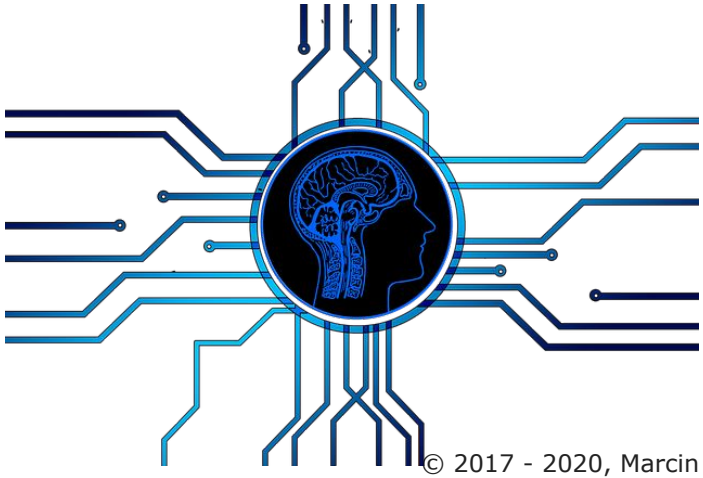
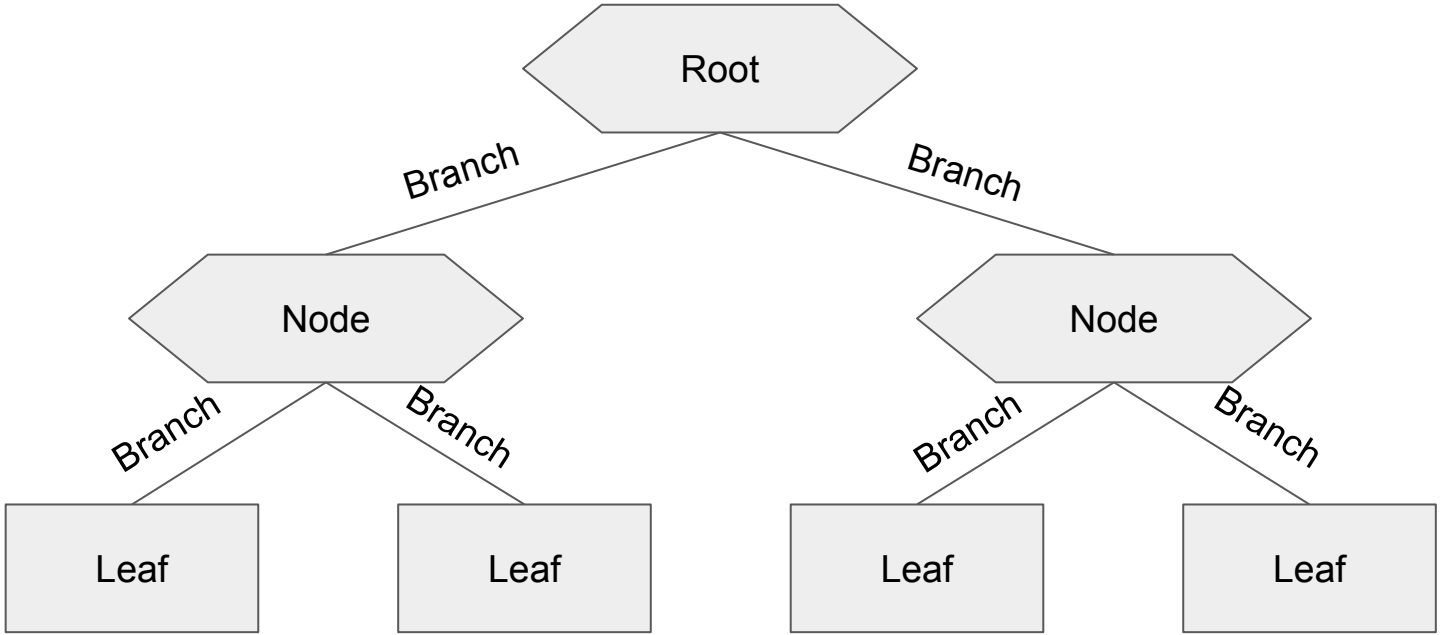
Machine Learning
Classical algorithms

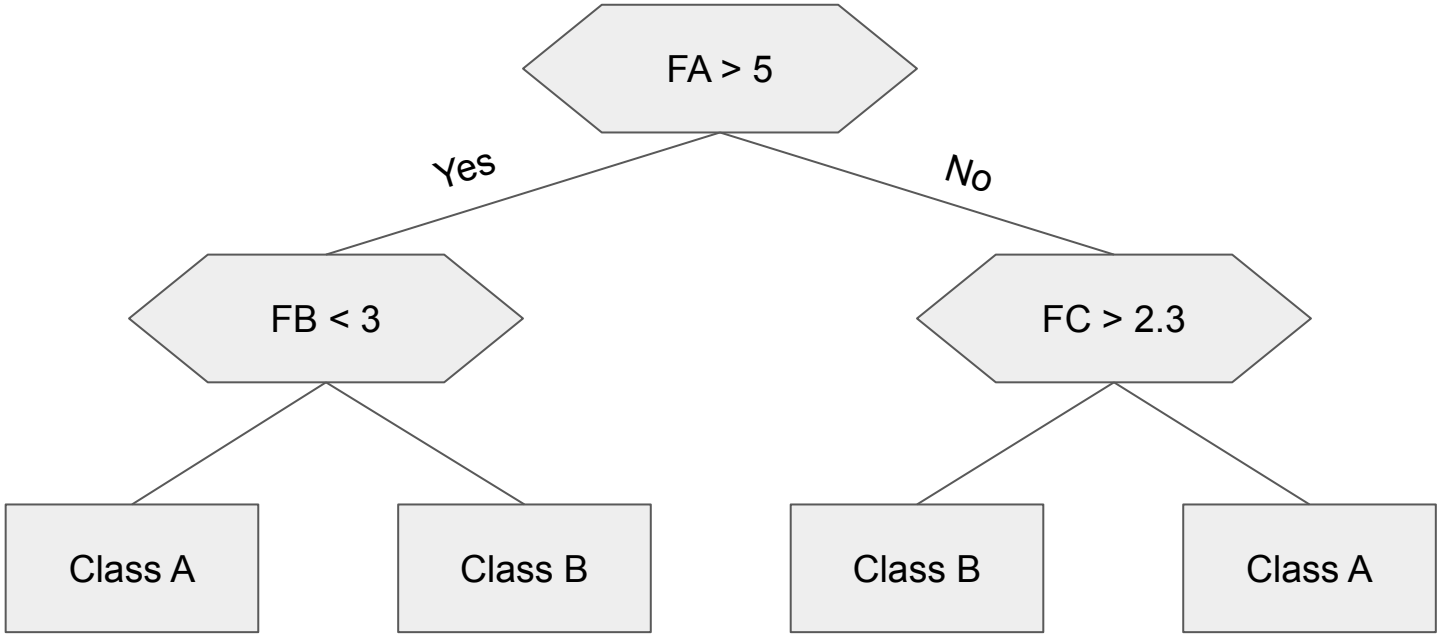


Decision Trees

Decision Tree Structure



Decision Tree Structure

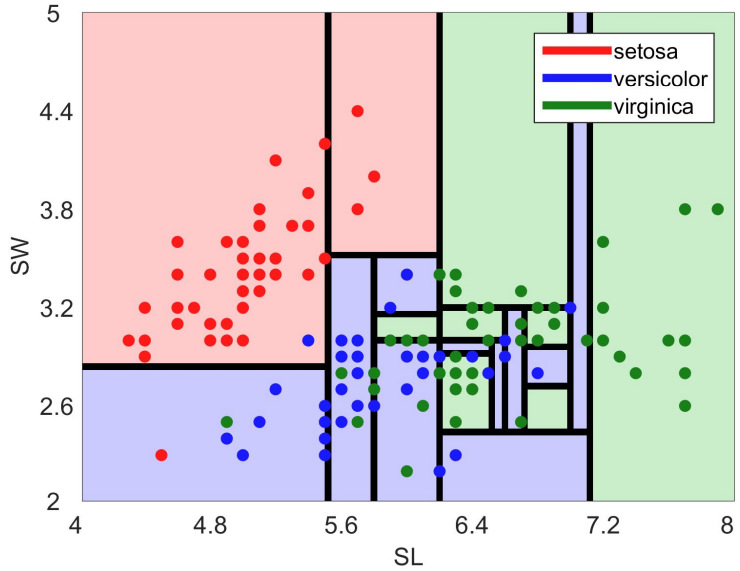


Algorithm

- Tree is grown from root
- Different algorithms exist but the process is usually as follows:
  1. Select feature that division of which will generate the most value
  2. Divide the feature
  3. Repeat the points 1-3 for each subset that was created until no further divisions possible
  4. Prune the tree, going from leafs up to the root check if branches can be removed
- There are different metrics that decide on the goodness of split
  - Information Gain
  - Gini Impurity (1 - p^2)
- The tree is pruned to avoid overfitting
  - Alternatives are providing max tree depth or min number of instances in a leaf

Decision Boundary

- Decision boundary of a decision tree is determined by overlapping orthogonal half-planes (each node creates a half-plane)
  - In 2D it looks like rectangles



Img src: https://www.mathworks.com/matlabcentral/fileexchange/52003-viewboundary

Naive Bayes

Bayes Theorem

- Bayes theorem states that:

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)},$$

- In ML we are dealing with similar problem:

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

$p(C_k \mid x)$ -> probability that an instance (x) belongs to a given class (C_k)

Naive Bayes

- Naive Bayes assumes features are independent
- Which simplifies equation to:

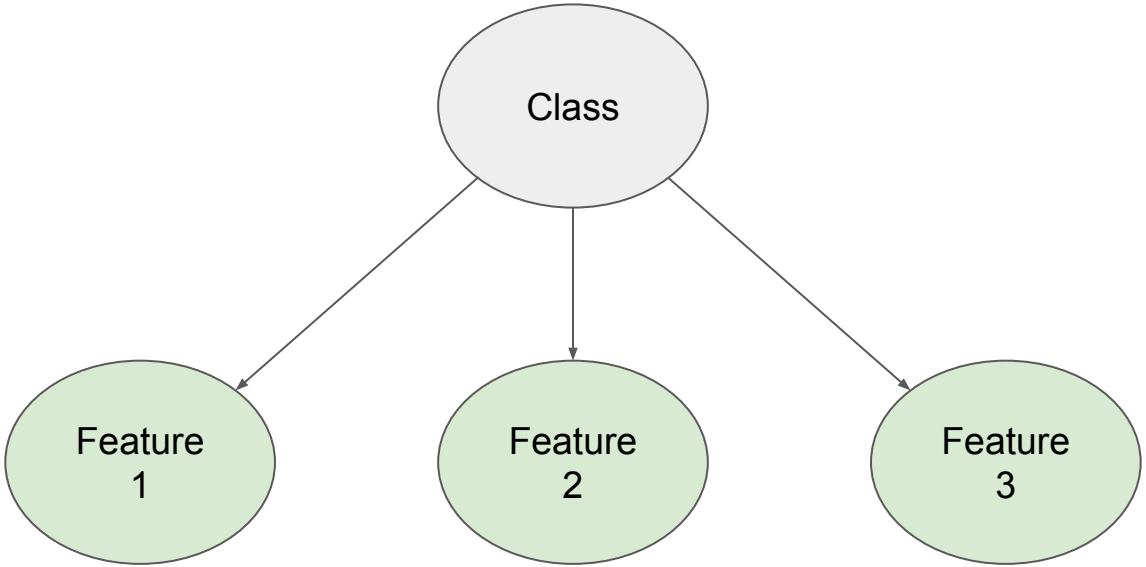
$$p(C_k \mid x_1, \ldots, x_n) = \frac{1}{Z} p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)$$

  - Where Z is a constant (probability of x)
- Per features probability is easy to calculate:
  - Count of occurences for nominal features
  - Discretization or gaussian approximation for continuous features
- To classify we need to select the class with the highest

$$p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)$$

Naive Bayes

- Assumption of features independence is most likely false
  - The classifier still performs quite strong despite of that
- It is very popular choice when dealing with text classification



Nearest Neighbours

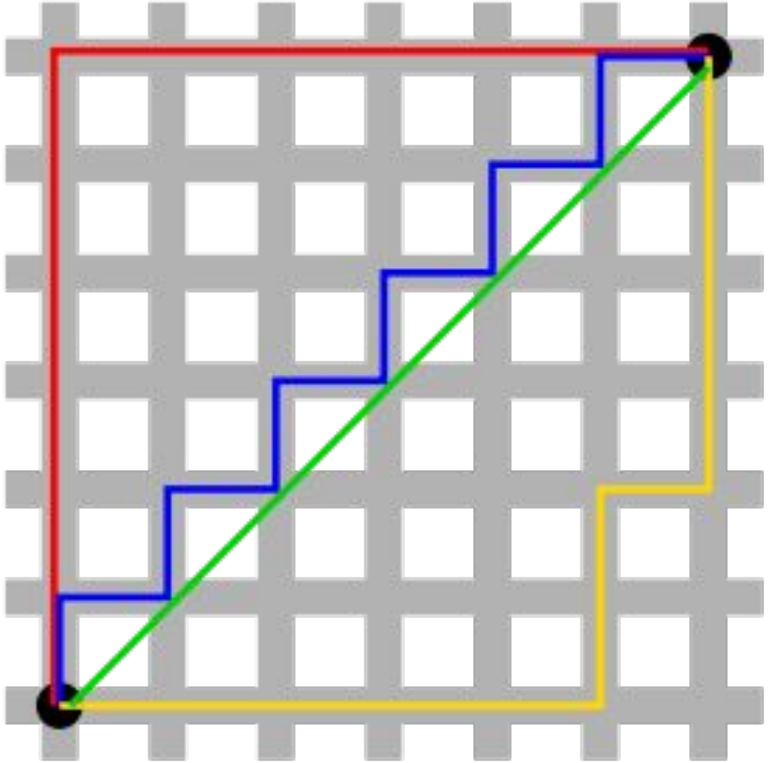
Nearest Neighbour

- Simple concept:
  - For an unseen instance find k closest instances (nearest neighbours) from training dataset
  - Select the most frequently occuring class among the k neighbours.
- This is a lazy evaluation algorithm
  - During training it just memorizes training dataset (whole dataset stored in memory)
  - Heavy computations are performed during evaluation phase (finding neighbours from all training samples)
- What defines nearest neighbours?

Distance Measures

- Euclidian
  - Measures absolute “straight line” distance between two points. Typically used metric
- Manhattan
  - May deal better with outliers and may be appropriate if different dimensions are not comparable.
- Cosine
  - Measures cosine of angle between two feature vectors (parallel = 0, orthogonal = 1)
  - Use when magnitude of vectors do not matter
- Hamming
  - Finds distance between all-binary instances
- … and many more. Which one to use?
  - Check different measures and use one that produces model with best performance

Distance Measures



Regression

Regression

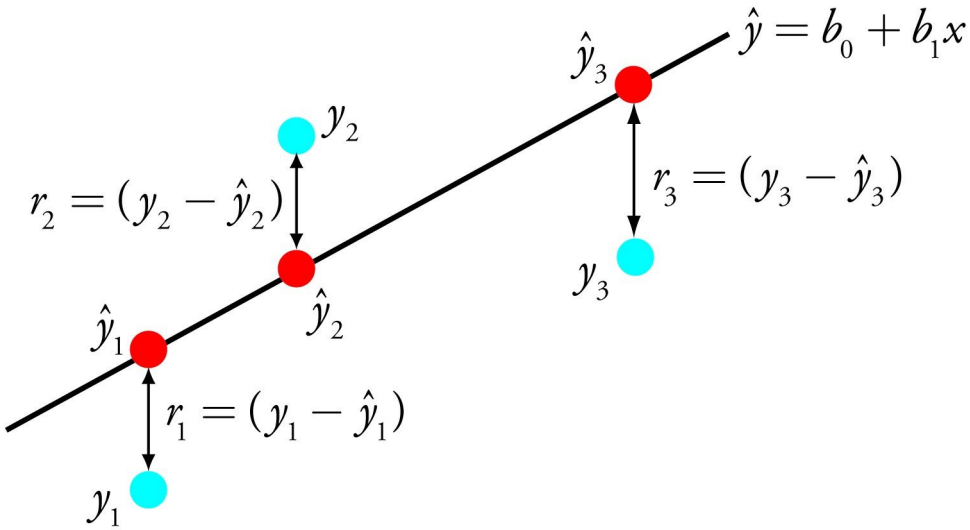
- ML can also be used to predict continuous target variables
- All three classifiers that we have learnt so far can be modified to do that
  - In DT each leaf will have mean value of instances
  - NN will return mean across neighbours
  - In NB target variable can be discretized
- But typically different algorithms are used for this task

Regression - Evaluation

- Target variable is continuous so there is no confusion matrix nor predicted probabilities
- MAE - Mean Absolute Error $\Sigma|Y(x_i) - Y'(x_i)|$
- MSE - Mean Squared Error $\Sigma(Y(x_i) - Y'(x_i))^2$
- R^2 - coefficient of determination
  - 1 is a perfect score
  - it can be less than 0

Linear regression

- Fit a line that will minimize MSE

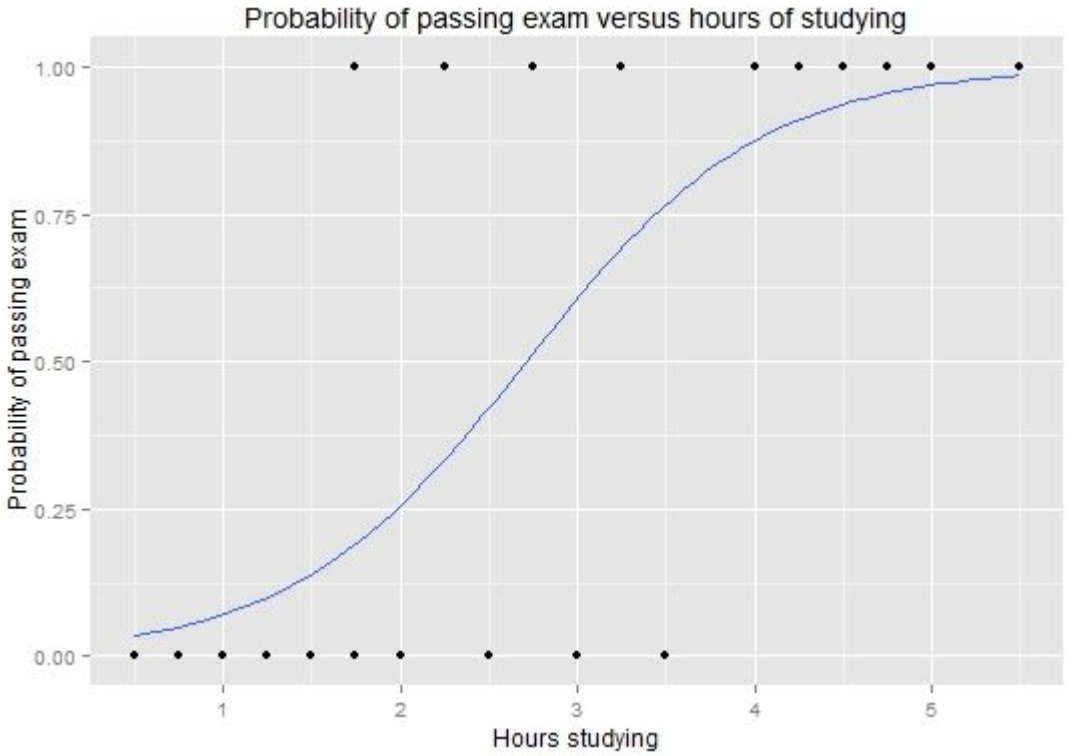


https://towardsdatascience.com/simple-linear-regression-2421076a5892

Logistic regression

- Regression models can also be used for classification problems
- Logistic regression is a good model when class is binary
- It is linear combination of features wrapped in a sigmoid activation function
  - Linear combination: $a = w_1x_1 + w_2x_2 + w_3x_3 + \ldots + b$
  - Sigmoid activation: $1/(1+e^{-a})$
- It can be seen as a simple neural network with one node
- Decision boundary is a hyperplane (line in 2D)
  - So it works great for classes that can be separated by straight line
  - In higher dimensions it’s more likely there will be a hyperplane that can separate classes

Logistic Regression - Example



What to do if there are more than 2 class values

- One-vs-Rest
  - Say you have k classes
  - For each class:
    - Train classifier predicting {class, not class}
  - Predict on test sample with all k classifiers
  - Each classifier gives probability [0, 1]
  - Pick the class with the classifier with highest prediction
- There are other techniques (one-vs-one, predicting distribution, etc.)

Regularization

- When building regression model it will tend to overfit to training set
- To prevent that we introduce a penalty for model complexity:
  - LossFunction(X,y) + α N(ω)
  - α - a scaling factor
  - N - a selected norm (L1, L2, or any other)
- Other classifiers also use regularization
  - Tree pruning (in general sense of preventing overfitting)
  - Distance metric

Regularization

- Examples from http://scikit-learn.org/stable/modules/linear_model.html:
  - Ridge Regression (L2-norm)

    $$\min_w ||Xw - y||_2{}^2 + \alpha||w||_2{}^2$$

    - Penalize sum of squared weights
    - This Penalty punishes more complex models
  - Lasso Regression (L1-norm)

    $$\min_w \frac{1}{2n_{samples}}||Xw - y||_2^2 + \alpha||w||_1$$

    - Penalizes absolute value of weights
    - Tends to favor sparse representations