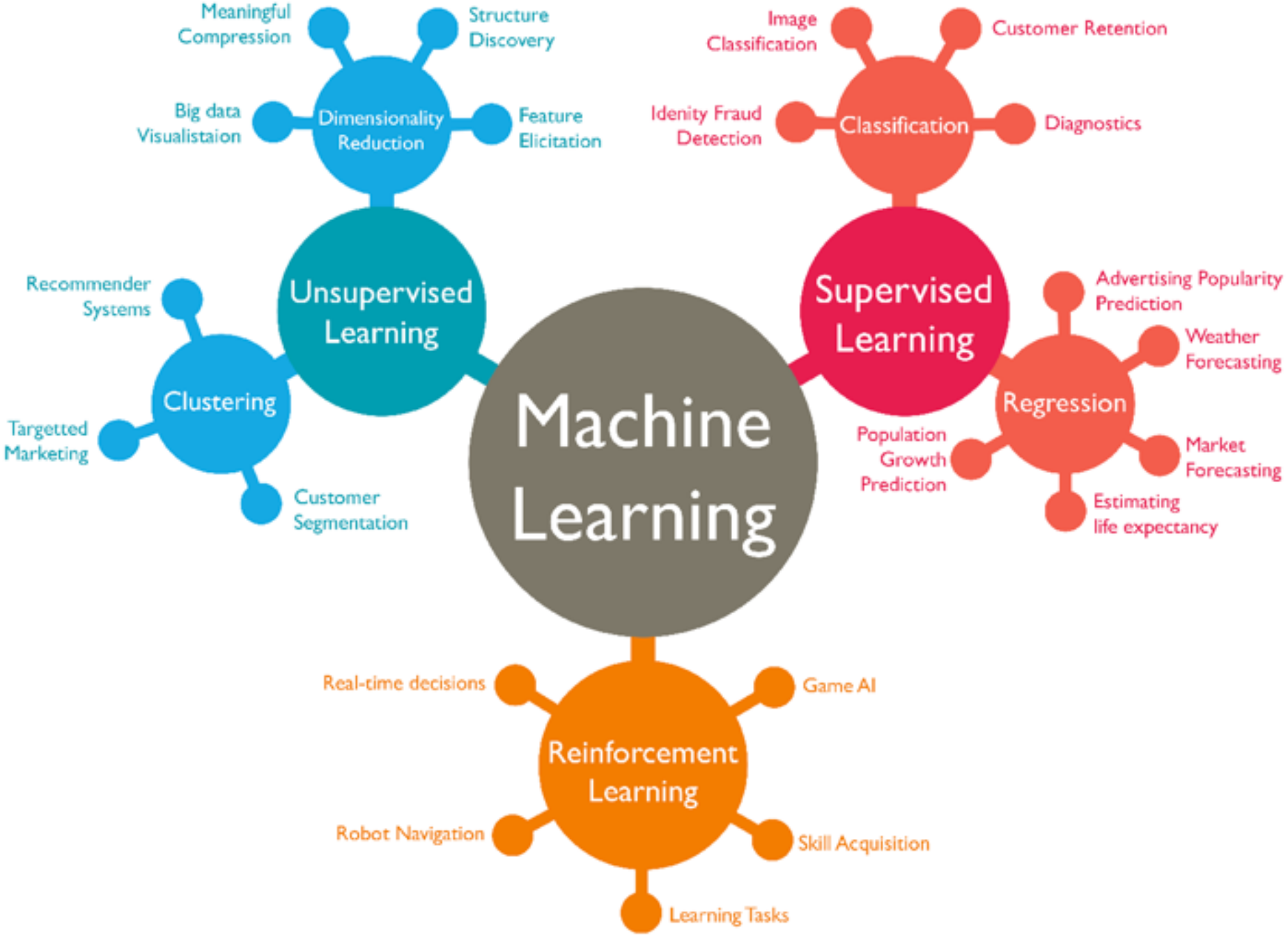
Supervised Learning: Classifications

Dr. Fabien Plisson

Chemoinformatics in Drug Discovery

LANGEBIO, UGA CINVESTAV

October 15-18, 2019 - Irapuato, Mexico


Machine Learning
Unsupervised Learning
Dimensionality Reduction
Meaningful Compression
Structure Discovery
Big data Visualistaion
Feature Elicitation
Clustering
Recommender Systems
Targetted Marketing
Customer Segmentation
Supervised Learning
Classification
Image Classification
Customer Retention
Idenity Fraud Detection
Diagnostics
Regression
Advertising Popularity Prediction
Weather Forecasting
Population Growth Prediction
Market Forecasting
Estimating life expectancy
Reinforcement Learning
Real-time decisions
Game AI
Robot Navigation
Skill Acquisition
Learning Tasks

Program

Classification problems & their metrics

Decision Tree

Ensemble methods

Support Vector Machine

K-Nearest Neighbour

Program

Classification problems & their metrics

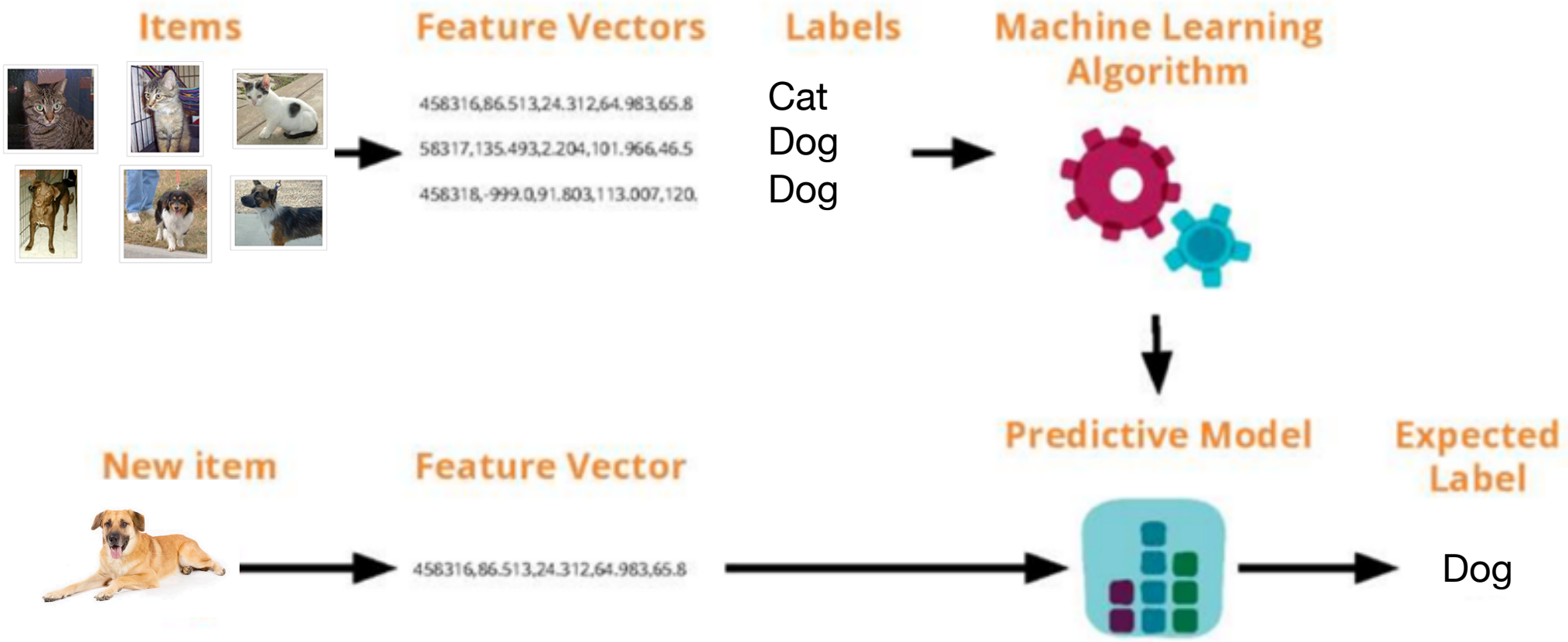
Decision Tree

Ensemble methods

Support Vector Machine

K-Nearest Neighbour

Classification problem



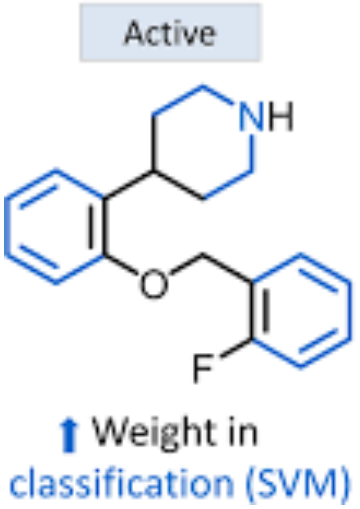
Classification problem

Assign the response into a **class**.

Predict new input to belong to one class or another.

Discrete / Categorical variable

Metric: **Accuracy**

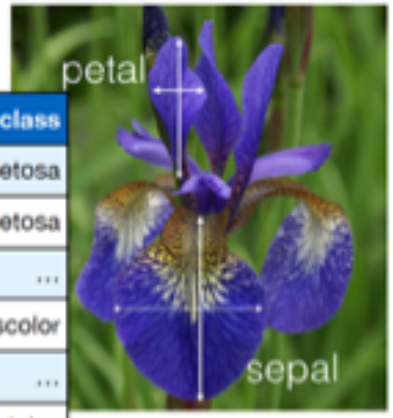


Nomenclature

IRIS

https://archive.ics.uci.edu/ml/datasets/Iris

Instances (samples, observations)



	sepal_length	sepal_width	petal_length	petal_width	class
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
...	...	...	...	...	...
50	6.4	3.2	4.5	1.5	veriscolor
...	...	...	...	...	...
150	5.9	3.0	5.1	1.8	virginica

Features (attributes, dimensions)

Classes (targets)

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ x_{31} & x_{32} & \cdots & x_{3D} \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix}$$

$$\mathbf{y} = [y_1, y_2, y_3, \cdots y_N]$$

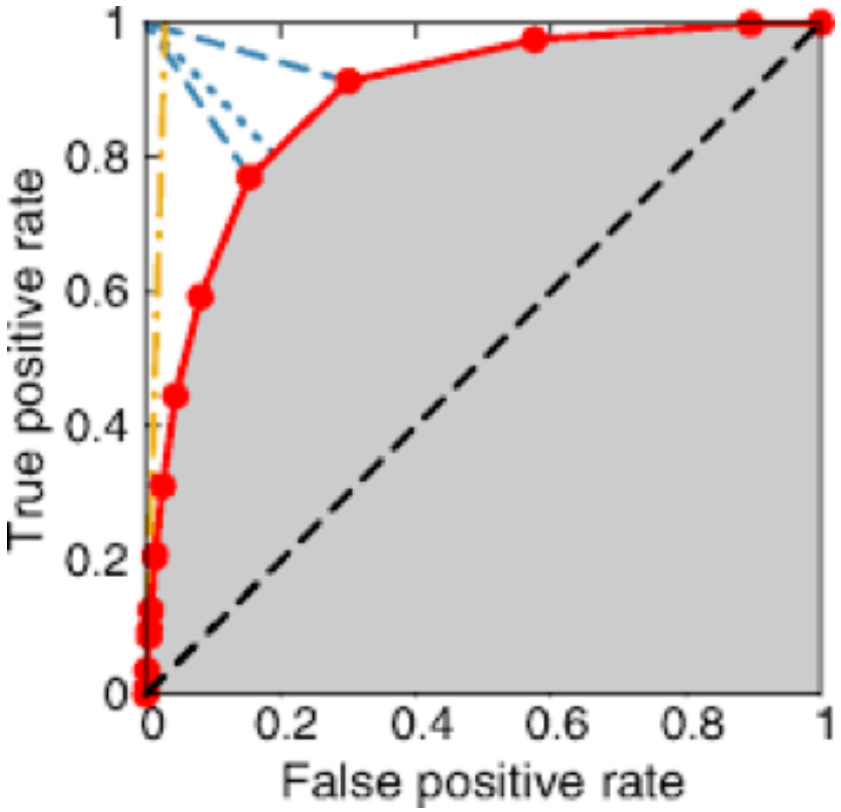
How do you measure performance of a classifier?
Confusion matrix

		Predicted class: P	Predicted class: N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

Accuracy (TN+TP)/(TN+FP+FN+TP)
Precision (TP)/(FP+TP)
Specificity (TN)/(FP+TN)
Sensitivity (TP)/(TP+FN)

Error Rate = 1 – Accuracy
Prevalence (FP+TP)/(TN+FP+FN+TP)

How do you measure performance of a classifier?
ROC Curve



True Positive Rate (TPR)
Sensitivity (TP)/(TP+FN)

False Positive Rate (FPR)
(1-Specificity) 1-**[**(TN)/(FP+TN)]

Scikit-learn: Classification Metrics

`metrics.accuracy_score` (y_true, y_pred[, …])	Accuracy classification score.
`metrics.auc` (x, y[, reorder])	Compute Area Under the Curve (AUC) using the trapezoidal rule
`metrics.average_precision_score` (y_true, y_score)	Compute average precision (AP) from prediction scores
`metrics.balanced_accuracy_score` (y_true, y_pred)	Compute the balanced accuracy
`metrics.brier_score_loss` (y_true, y_prob[, …])	Compute the Brier score.
`metrics.classification_report` (y_true, y_pred)	Build a text report showing the main classification metrics
`metrics.cohen_kappa_score` (y1, y2[, labels, …])	Cohen's kappa: a statistic that measures inter-annotator agreement.
`metrics.confusion_matrix` (y_true, y_pred[, …])	Compute confusion matrix to evaluate the accuracy of a classification
`metrics.f1_score` (y_true, y_pred[, labels, …])	Compute the F1 score, also known as balanced F-score or F-measure
`metrics.fbeta_score` (y_true, y_pred, beta[, …])	Compute the F-beta score
`metrics.hamming_loss` (y_true, y_pred[, …])	Compute the average Hamming loss.
`metrics.hinge_loss` (y_true, pred_decision[, …])	Average hinge loss (non-regularized)
`metrics.jaccard_score` (y_true, y_pred[, …])	Jaccard similarity coefficient score
`metrics.log_loss` (y_true, y_pred[, eps, …])	Log loss, aka logistic loss or cross-entropy loss.
`metrics.matthews_corrcoef` (y_true, y_pred[, …])	Compute the Matthews correlation coefficient (MCC)
`metrics.multilabel_confusion_matrix` (y_true, …)	Compute a confusion matrix for each class or sample
`metrics.precision_recall_curve` (y_true, …)	Compute precision-recall pairs for different probability thresholds
`metrics.precision_recall_fscore_support` (…)	Compute precision, recall, F-measure and support for each class
`metrics.precision_score` (y_true, y_pred[, …])	Compute the precision
`metrics.recall_score` (y_true, y_pred[, …])	Compute the recall
`metrics.roc_auc_score` (y_true, y_score[, …])	Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.
`metrics.roc_curve` (y_true, y_score[, …])	Compute Receiver operating characteristic (ROC)
`metrics.zero_one_loss` (y_true, y_pred[, …])	Zero-one classification loss.

https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics

Program

Classification problems & their metrics

Decision Tree

Ensemble methods

Support Vector Machine

K-Nearest Neighbour

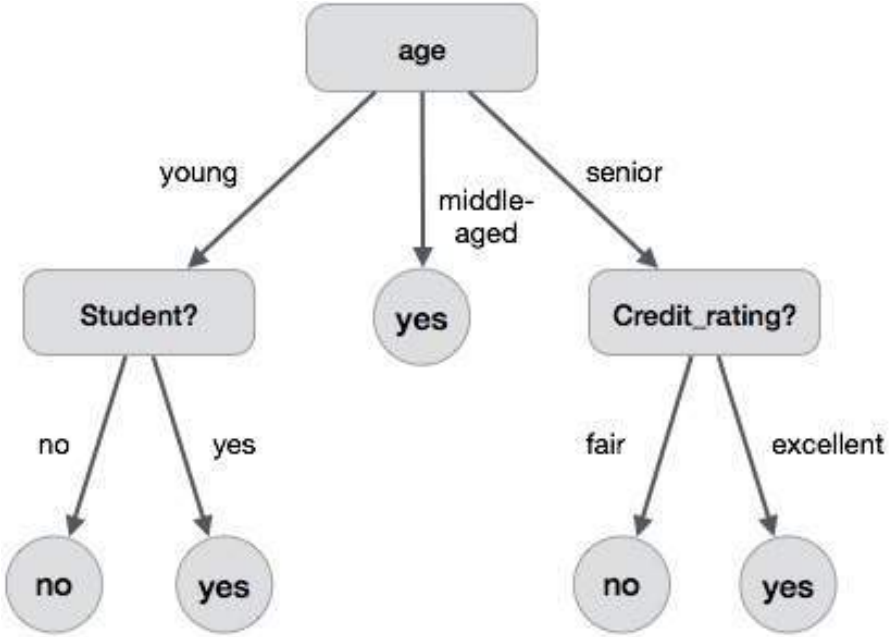
Decision Tree (CART)

Graphical representation of possible solutions based on certain conditions

Root Node = Top Node / Best predictor (feature)

Decision / Internal Node = the nodes where predictors (features) are tested, each branch represents an outcome of the test

Leaf / Terminal Node = It holds a class label (category) e.g. yes or no

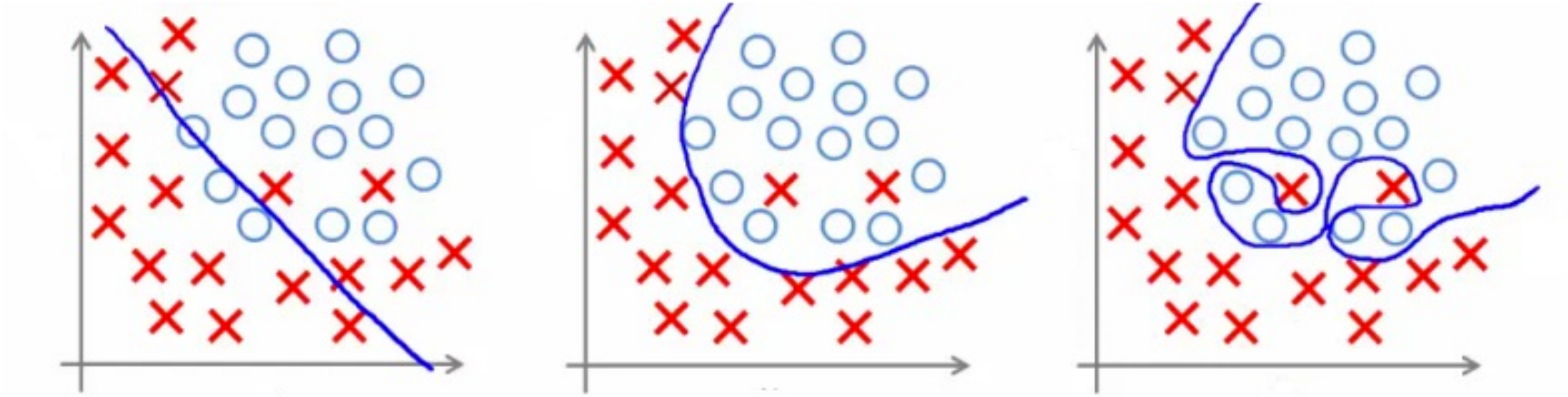


Pros and Cons

Easy to interpret
Works even if the relationships between features are non-linear
Not sensitive to outliers

Decision Tree model generally **overfits**
Assumes all features (independent variables) interact

Overfitting



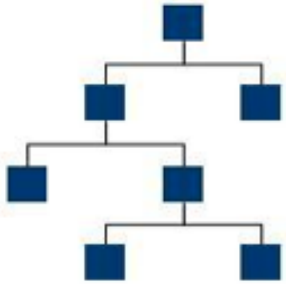
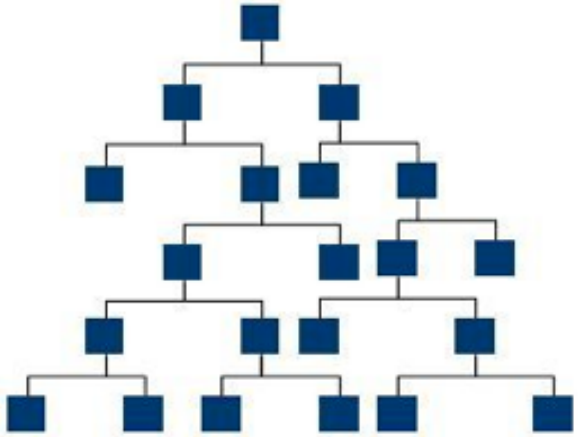
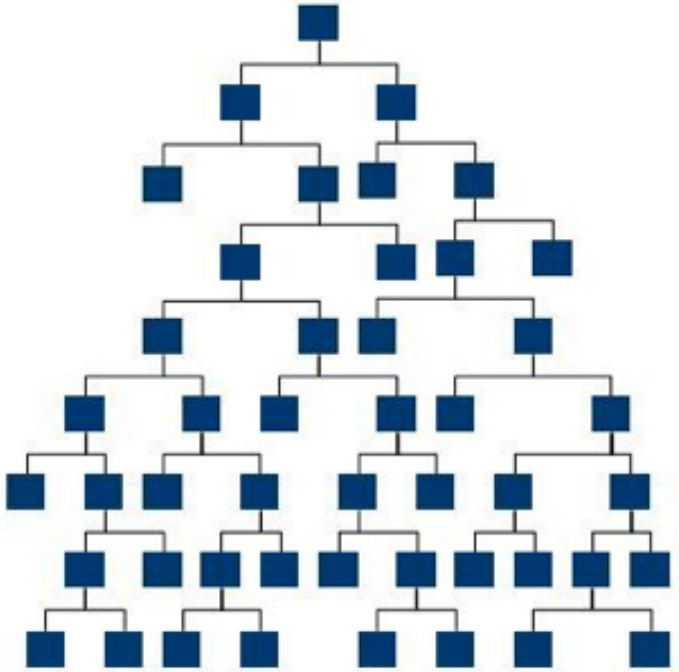
Under-fitting

(too simple to explain the variance)

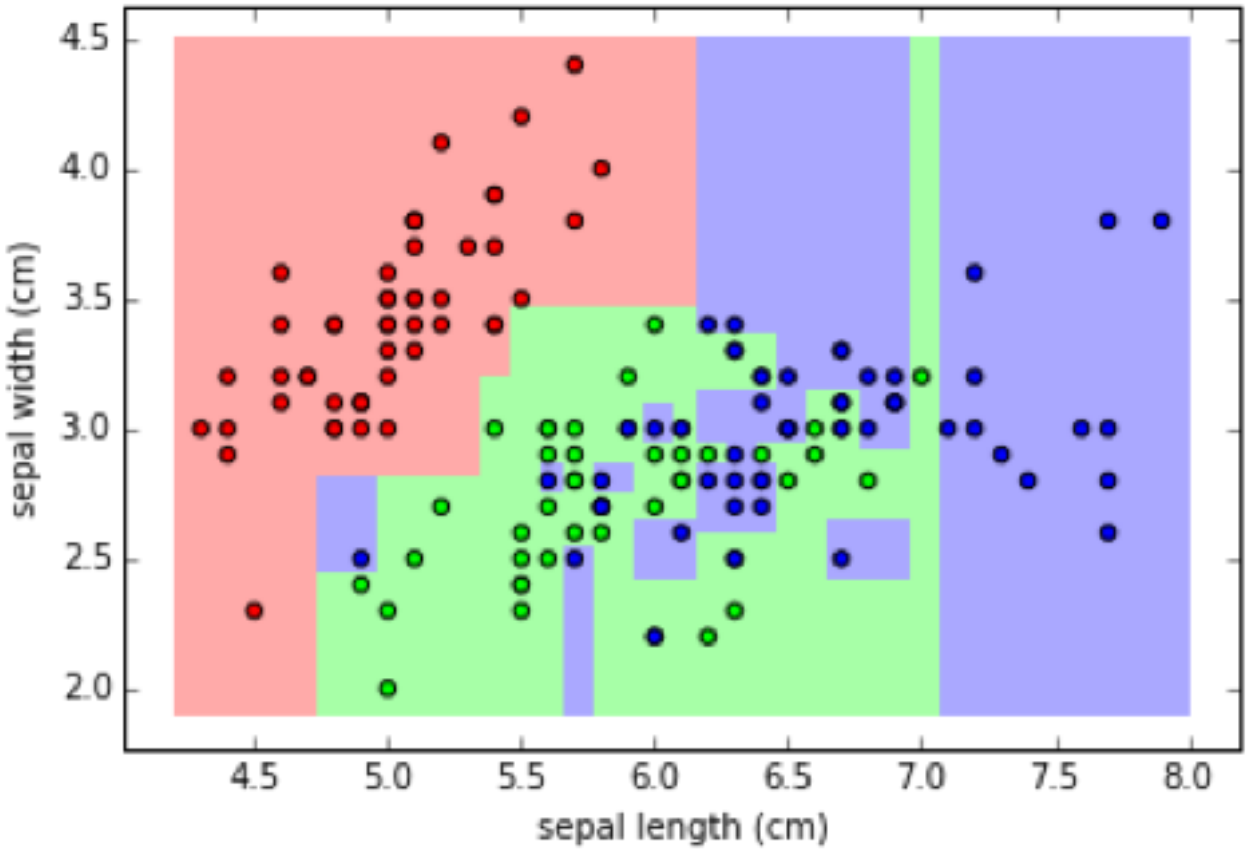
Appropriate-fitting

Over-fitting

(forcefitting -- too good to be true)

Underfit tree	Optimal tree	Overfit tree
		
Accuracy on training = 50%	Accuracy on training = 70%	Accuracy on training = 90%
Accuracy on test = 50%	Accuracy on test = 70%	Accuracy on test = 65%

Overfitted Decision Tree



Pruning

Correct overfitting by reducing the size of decision trees and reducing features that have little predictive power to classify instances (observations) > LESS COMPLEX TREES

Early Stopping/Pre-Pruning

No significant association (**chi-square test**) between individual feature(s) and the classes

Faster than Post-pruning method

Post-Pruning

Build full tree & prune it using various methods – **cost complexity (CP)** – reduce number of leaves & measure error rate e.g. sum-of-squares error

How does the classification work?

1. Pick the feature that gives the best split → **Gini Index** / **Entropy / Information Gain**
2. Partition the data based on the value of this feature.

3. Repeat steps 1 and 2.
4. Splitting stops when;

- Decision tree detects no further gain can be made,
- Early stopping are met,
- The data is splitter as much as possible and the tree is later pruned.

Splitting criteria

Gini Index (~ impurity)

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

Gini = 0 perfect classification
GI = (1-(1/N)) worst classification
Feature split on low GI
On binary classification, GI <= 0.5

Entropy / Information Gain

$$Entropy = \sum_{i=1}^{C} -p_i * \log_2(p_i)$$

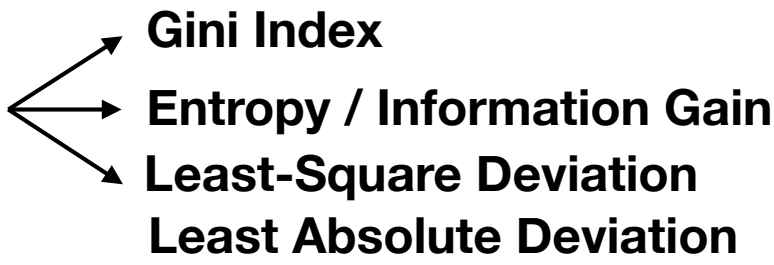
Favours partitions with many distinct values

Smaller Entropy = good classification

Similar results in general, Gini Index faster to execute

How does the regression work?

1. Pick the feature that gives the best split



2. Partition the data based on the value of this feature.

3. Repeat steps 1 and 2.
4. Splitting stops when;

- Decision tree detects no further gain can be made,
- Early stopping are met,
- The data is splitter as much as possible and the tree is later pruned.

Animated Explanation

http://www.r2d3.us

Program

Classification problems & their metrics

Decision Tree

Ensemble methods

Support Vector Machine

K-Nearest Neighbour

Ensemble Methods

Ensemble methods combine the decisions from multiple predictive models to improve the overall performance (better accuracy) and model stability (reduce bias, variance and noise)

Simple methods:
Mode, Average, Weighted Average

Advanced methods:
Bagging – reduce variance **Random Forest**
Boosting – reduce bias **Gradient Boosting**
Stacking

Reducing predictive error

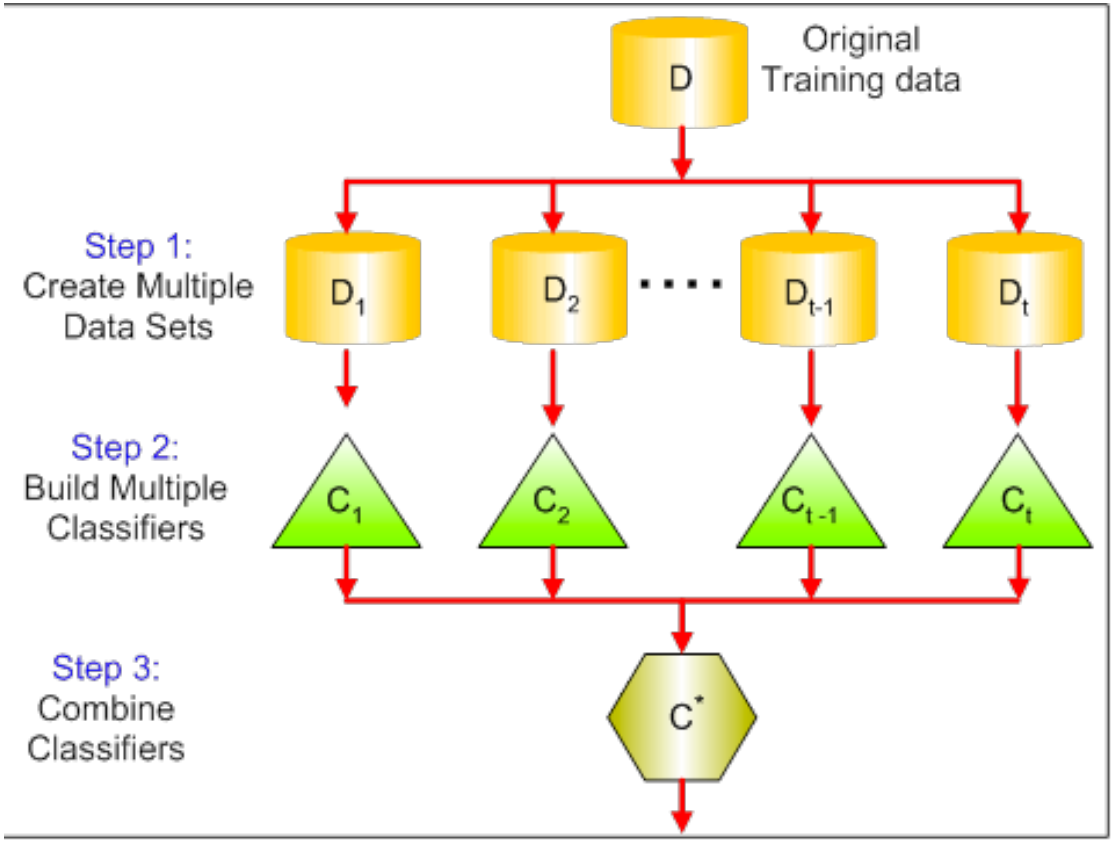
Total (Predictive) Error $$Err(x) = \left(E[\hat{f}\,(x)] - f(x)\right)^2 + E\left[\left(\hat{f}\,(x) - E[\hat{f}\,(x)]\right)^2\right] + \sigma_e^2$$

$$Err(x) = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.

Variance is the variability of model prediction for a given data point or a value which tells us spread of our data. Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn't seen before. As a result, such models perform very well on training data but has high error rates on test data.

Bagging - Boostrap Aggregating



1- Create training subsets **randomly with replacement** = Boostrap samples or Bags

2- Build models based on these boostrap samples

3- Combine classifiers and calculate **average or majority voting**

How does Random Forest work?

Random Forest boostrap aggregating multiple CART models

1- Assume number *N* of observations in training set

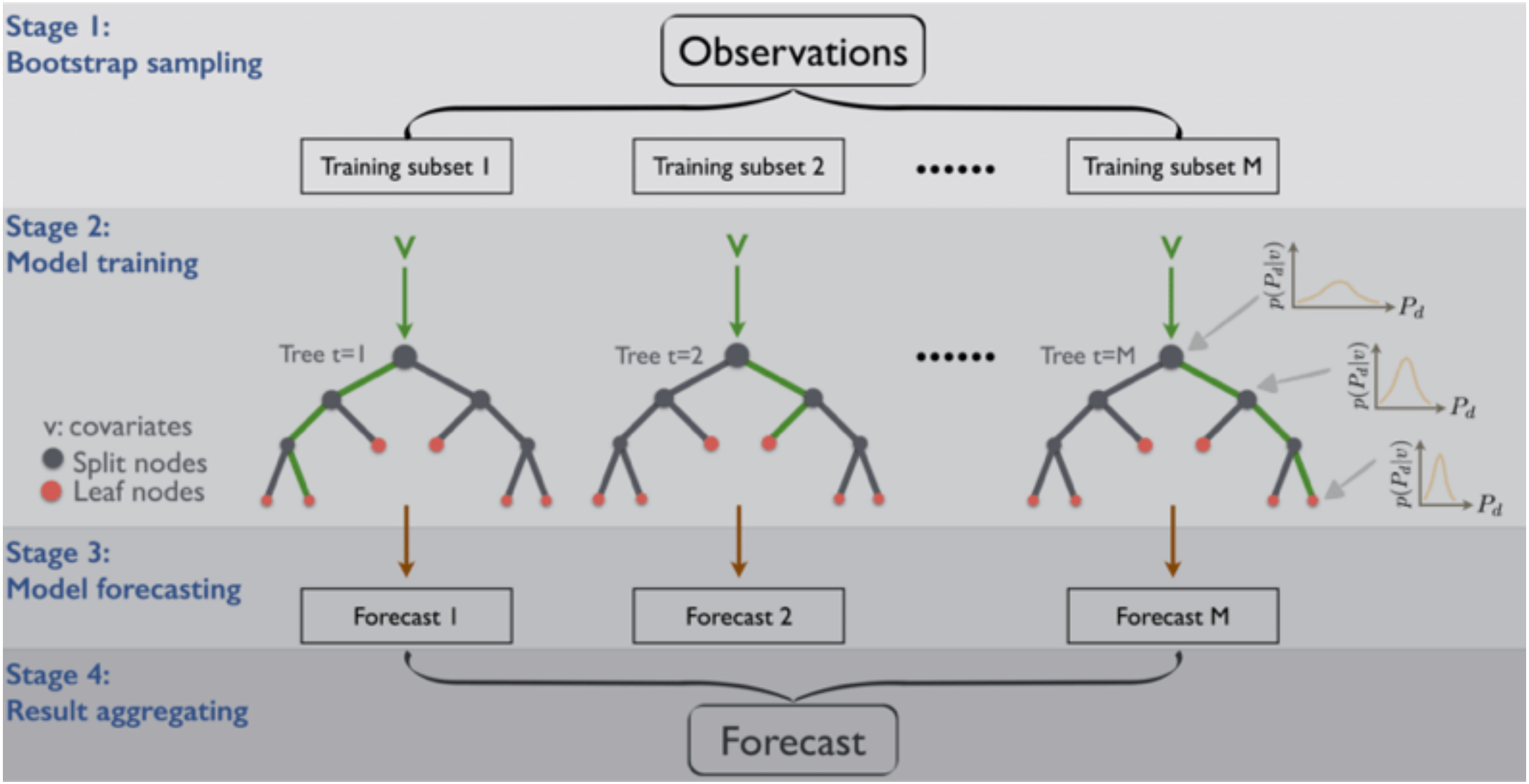
2- A sample of these *N* observations is taken **randomly with replacement** = new training set

3- If there are *M* input features, a number $m<M$ is specified at each node. *m* are features selected at random out of the *M*. **The best split on these *m* features is used to split the node**. The value of *m* is held constant while we grow the forest.

4- Each tree is grown to the largest extent possible and there is **no pruning**.

5- Predict new data by aggregating the predictions of the trees (i.e. majority votes for classification, average for regression)

How does Random Forest work?

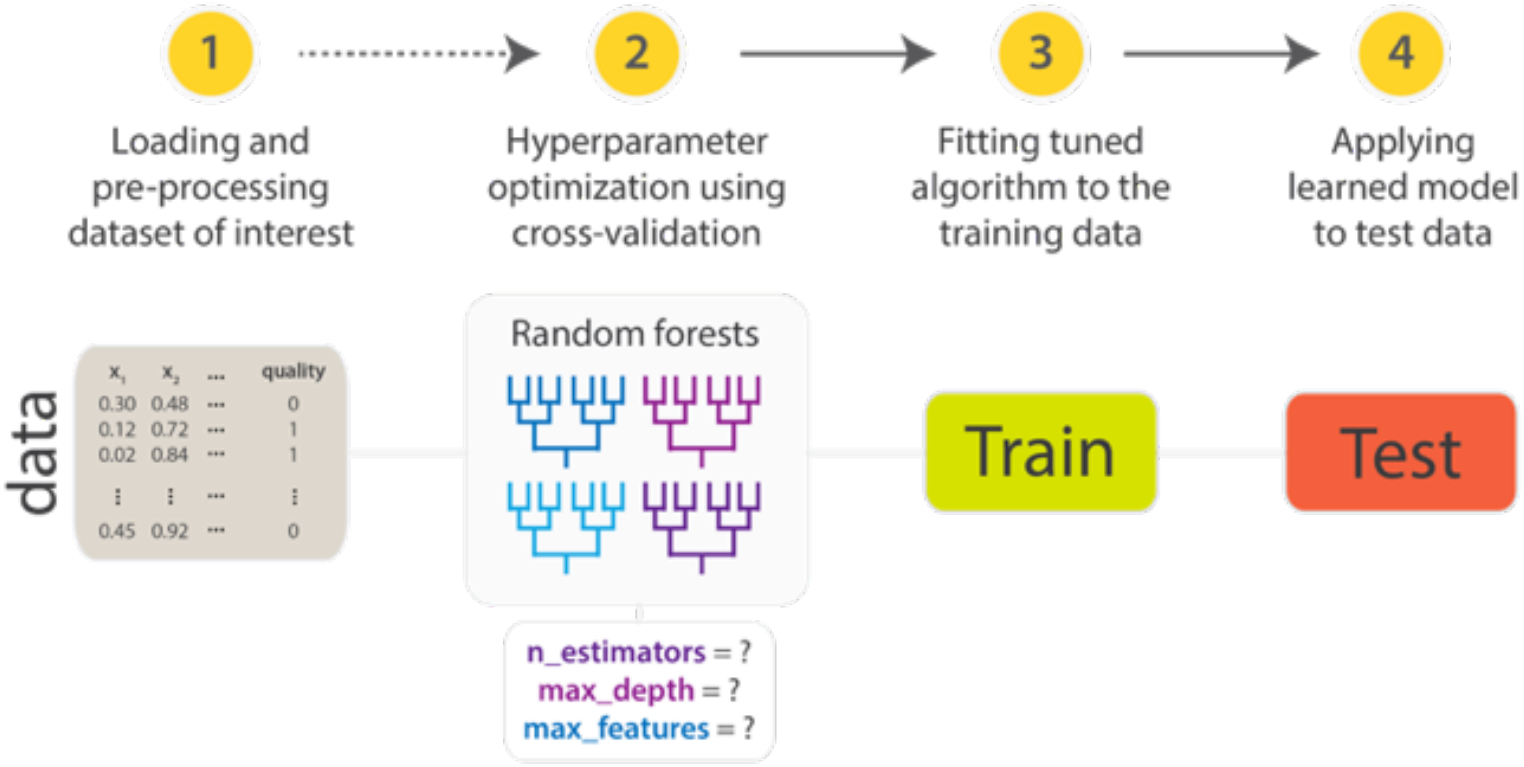


Pros and Cons

- Used for classification and regression modelling
- Handles large datasets with high dimensionality
- Outputs features in order of importance (**Feature Importance**)
  > Dimension Reduction
- Can estimate missing data values, maintain accuracy in absence of data
- Can balance errors when dataset is imbalanced
- Build models on training sets (boostrapped trees “bags”) while testing set = **out of bag** samples. Error estimated on these out of bag samples is known as **out of bag error**. Study of OOB errors is a performance metric for testing set = training set.

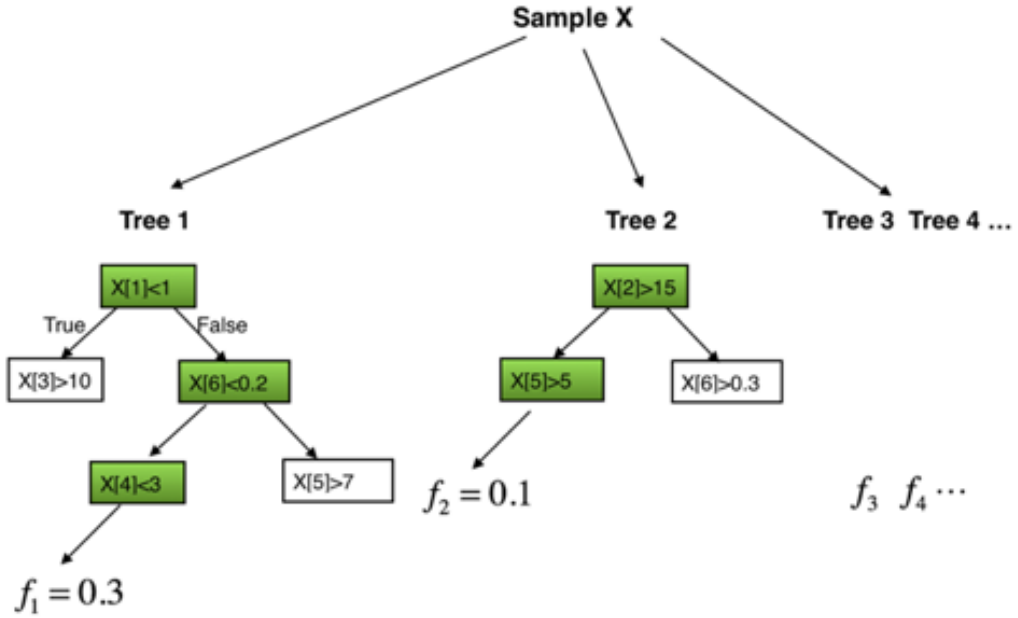
- In a regression model, RF does not predict beyond label values out of training set
- Limited control on RF modelling (tuning hyper-parameters and random seeds)

Tuning (Hyper-)Parameters



Number of trees **n_estimators**
Maximum depth **max_depth**
Maximum features per split **max_features**
Minimum number of samples required to be at a leaf node of a tree **min_samples_leaf**

Boosting



Transform features considered as **weaker learners** into stronger learners using **average/weighted averages or majority vote**

Bagging*:* independent bags of equal weight

Boosting*:* sequential process where the first model is trained on the entire dataset and subsequent models are built by fitting the residuals of the first model (reducible error) , thus giving higher weight to those observations that were poorly predicted by the previous model.

How does Gradient Boosting work?

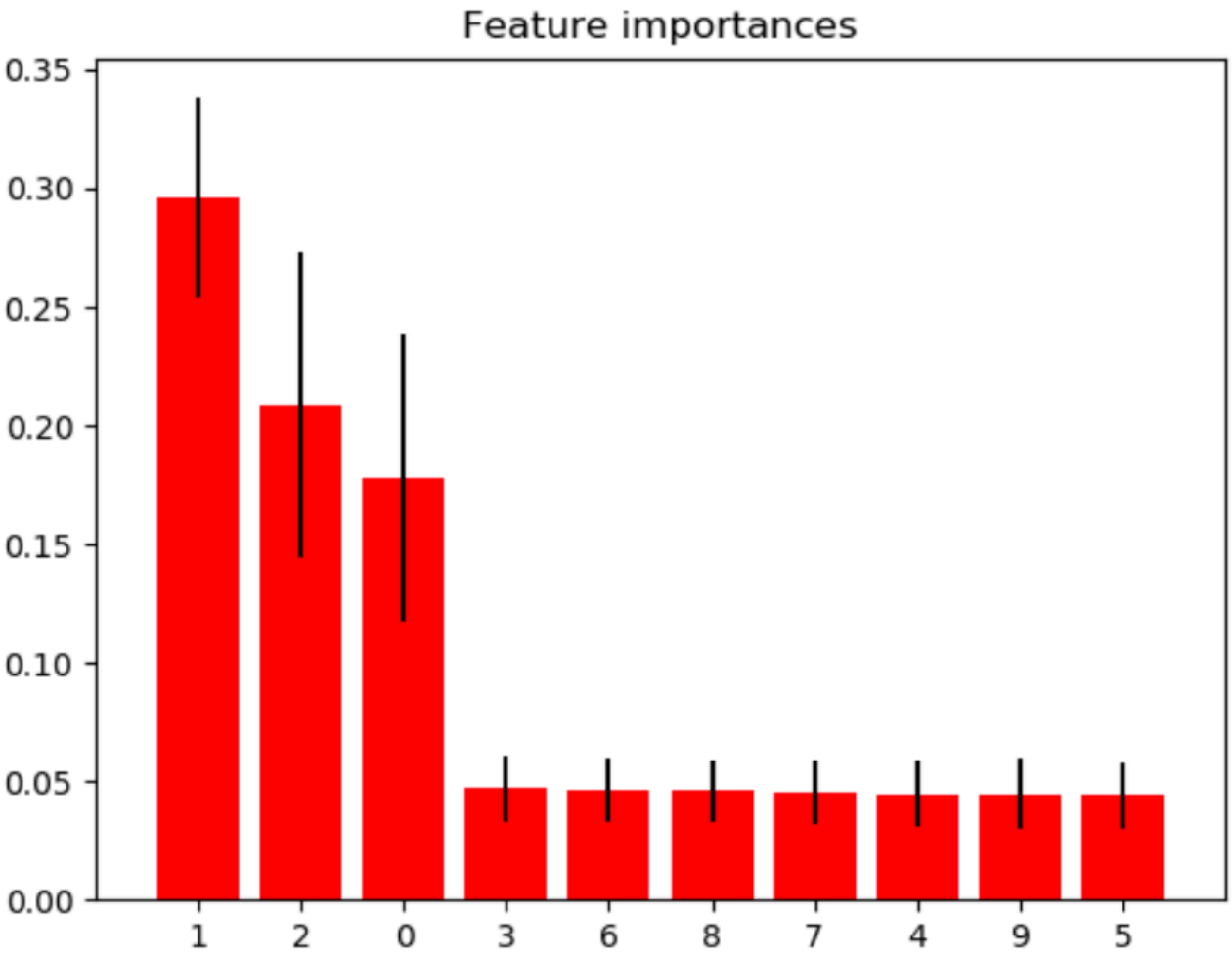
Gradient Boosting Machine Boosting multiple CART models

1. Initialise the outcome

2. Iterate from 1 to N , total number of trees
   2.1 Update the weights for targets based on previous run (misclassified observations)
   2.2 Fit the model on selected subsample of data
   2.3 Make predictions on the full set of observations
   2.4 Update the output with current results taking into account the learning rate

3. Return the final output.

Feature Importance




```
Feature ranking:
1. feature 1 (0.295902)
2. feature 2 (0.208351)
3. feature 0 (0.177632)
4. feature 3 (0.047121)
5. feature 6 (0.046303)
6. feature 8 (0.046013)
7. feature 7 (0.045575)
8. feature 4 (0.044614)
9. feature 9 (0.044577)
10. feature 5 (0.043912)
```

Online Lessons

Trevor Hastie - Gradient Boosting Machine Learning
https://www.youtube.com/watch?v=wPqtzj5VZus

Tianqi Chen - XGBoost A Scalable Tree Boosting System
https://www.youtube.com/watch?time_continue=1627&v=Vly8xGnNiWs

Tong He – XGBoost: eXtreme Gradient Boosting
https://www.youtube.com/watch?time_continue=2169&v=ufHo8vbk6g4

`sklearn.ensemble`: Ensemble Methods

The `sklearn.ensemble` module includes ensemble-based methods for classification, regression and anomaly detection.

User guide: See the Ensemble methods section for further details.

`ensemble.AdaBoostClassifier` ([…])	An AdaBoost classifier.
`ensemble.AdaBoostRegressor` ([base_estimator, …])	An AdaBoost regressor.
`ensemble.BaggingClassifier` ([base_estimator, …])	A Bagging classifier.
`ensemble.BaggingRegressor` ([base_estimator, …])	A Bagging regressor.
`ensemble.ExtraTreesClassifier` ([…])	An extra-trees classifier.
`ensemble.ExtraTreesRegressor` ([n_estimators, …])	An extra-trees regressor.
`ensemble.GradientBoostingClassifier` ([loss, …])	Gradient Boosting for classification.
`ensemble.GradientBoostingRegressor` ([loss, …])	Gradient Boosting for regression.
`ensemble.IsolationForest` ([n_estimators, …])	Isolation Forest Algorithm
`ensemble.RandomForestClassifier` ([…])	A random forest classifier.
`ensemble.RandomForestRegressor` ([…])	A random forest regressor.
`ensemble.RandomTreesEmbedding` ([…])	An ensemble of totally random trees.
`ensemble.VotingClassifier` (estimators[, …])	Soft Voting/Majority Rule classifier for unfitted estimators.
`ensemble.VotingRegressor` (estimators[, …])	Prediction voting regressor for unfitted estimators.
`ensemble.HistGradientBoostingRegressor` ([…])	Histogram-based Gradient Boosting Regression Tree.
`ensemble.HistGradientBoostingClassifier` ([…])	Histogram-based Gradient Boosting Classification Tree.

https://scikit-learn.org/stable/modules/ensemble.html

Program

Classification problems & their metrics

Decision Tree

Ensemble methods

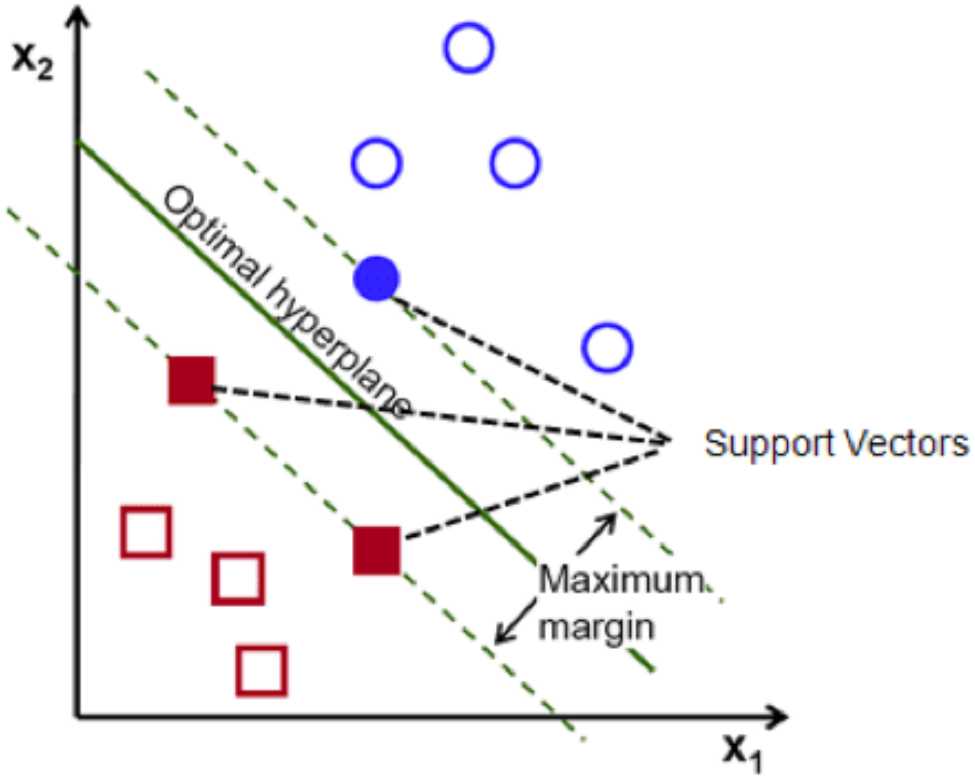
Support Vector Machine

K-Nearest Neighbour

Support Vector Machine (SVM)

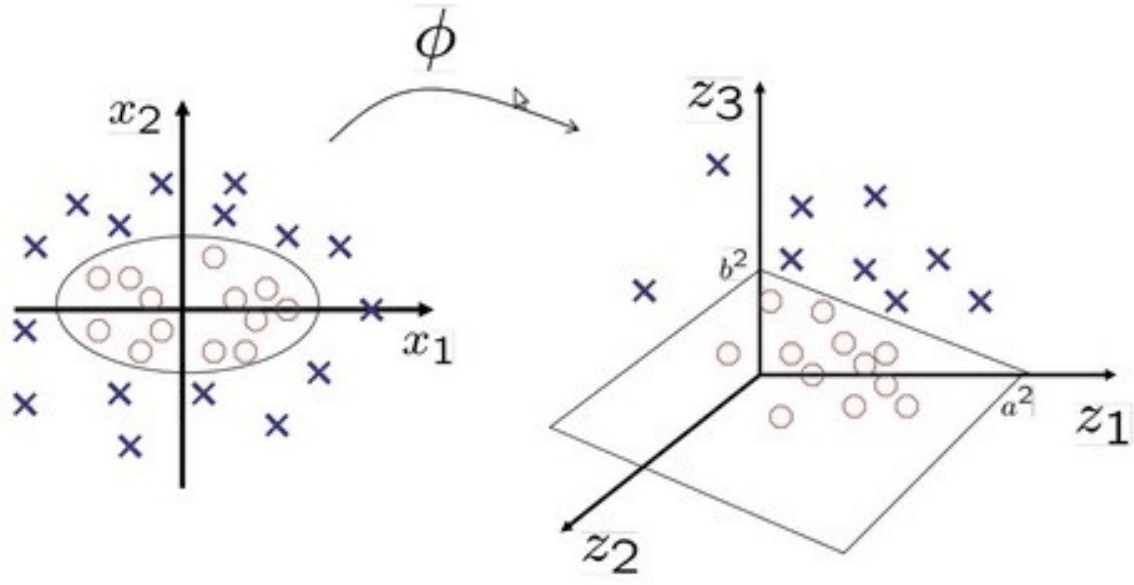
Support Vector Machine (SVM) = finding the **optimal hyperplane (**line in 2D, plane in 3D, hyperplane in >3D) which **maximizes the margin between two classes.**

Support vectors = are observations that supports hyperplane on either sides



Kernel Function

= method to make SVM run in case of non-linear separable data points. The kernel function transforms the data into a higher dimensional feature space to make it possible to perform the linear separation.



$$\phi : (x_1, x_2) \longrightarrow (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

Kernel Functions

- linear: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$.
- polynomial: $K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma {\mathbf{x}_i}^T \mathbf{x}_j + r)^d$, $\gamma > 0$.
- radial basis function (RBF): $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$, $\gamma > 0$.
- sigmoid: $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma {\mathbf{x}_i}^T \mathbf{x}_j + r)$.

How does SVM work?

1- Choose an optimal hyperplane which maximize margin

2- Applies penalty for misclassification (**cost 'c' tuning parameter**).

3- If non-linearly separable data points, transform data to high dimensional space where it is easier to classify with linear decision surfaces (**Kernel trick**)

Pros and Cons

Applicable to classifications and regressions
Performs well in case of non-linear separable data using kernel trick
Works well in higher dimensional space e.g. image/text classification
Does not suffer from multicollinearity
Can be applied to multi-category classifications (>2) using *one-versus-one or one-versus-rest binary classes.*

Time consuming on large datasets
Does not directly return probability estimates
Linear kernel ~ logistic regression in case of separable linear data

SVM Standardisation

Kernel methods are based on distance requires **scaling data** using Z-score or Min-Max

Kernel values usually depend on the inner products of feature vectors, e.g. the linear kernel and the polynomial kernel, large attribute values might cause numerical problems.

min-max normalization

$$v' = \frac{v - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A$$

z-score normalization

$$v' = \frac{v - mean_A}{stand\_dev_A}$$

Tuning Kernel (Hyper-)Parameters

Linear Kernel

Cost C (cost of misclassification)

large C = low bias/high variance, potentially overfits

Radial Basis Function (RBF)

Cost C, Gamma g (importance of training)

small g = *model too constrained that cannot capture the complexity of the data*

e.g. cost=10^(-1:2), gamma=c(0.5, 1, 2)

Polynomial

Degree (Polynomial degree),
scale (Scale), Cost C

SV Regression

Epsilon e and Cost C

Program

Classification problems & their metrics

Decision Tree

Ensemble methods

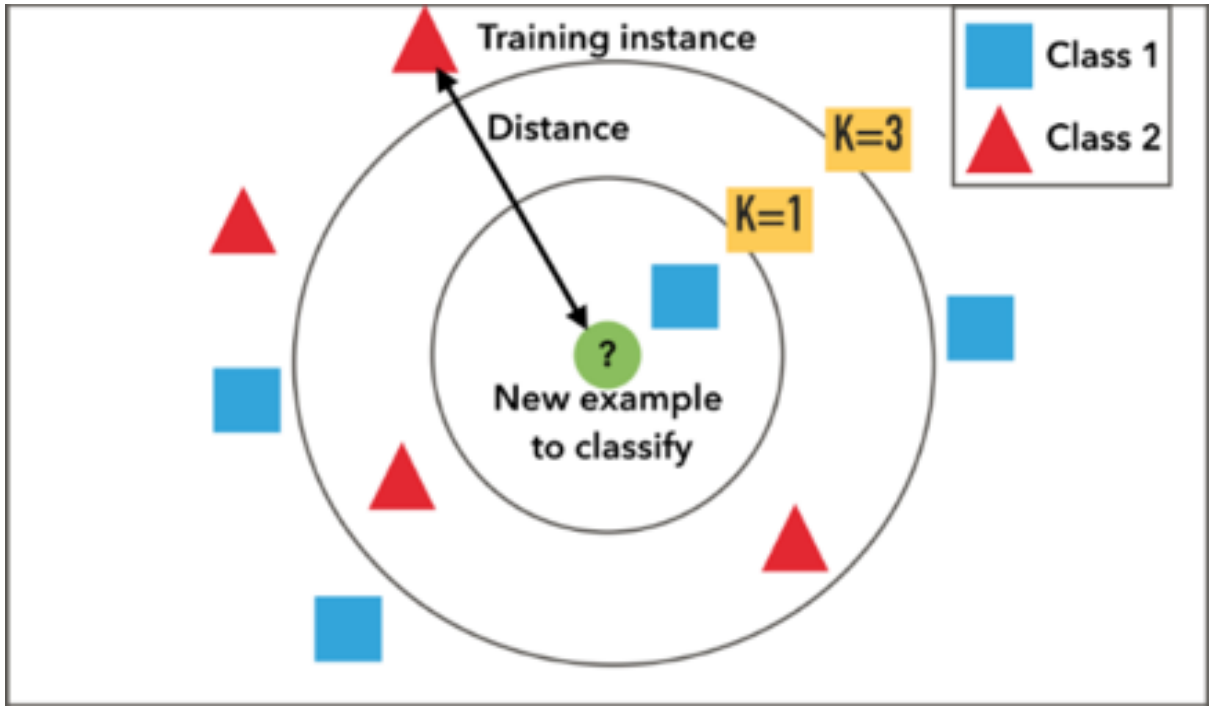
Support Vector Machine

K-Nearest Neighbour

K-Nearest Neighbour (K-NN)

K-Nearest Neighbour is **non-parametric** algorithm where we classify the data point to a given category with the help of training set, based on a **feature similarity.**

Predictions are made for a new instance (x) by searching through the entire training set for the K most similar cases (neighbours) and summarising the output variable for those K cases. In classification this is the mode (or most common) class value.



How does K-NN work?

1. Calculate the similarity (distance) metrics between observations
2. Find K nearest observations (sharing similar features) – training set
3. Based on K, assign the class to new data based on majority vote

Assumptions

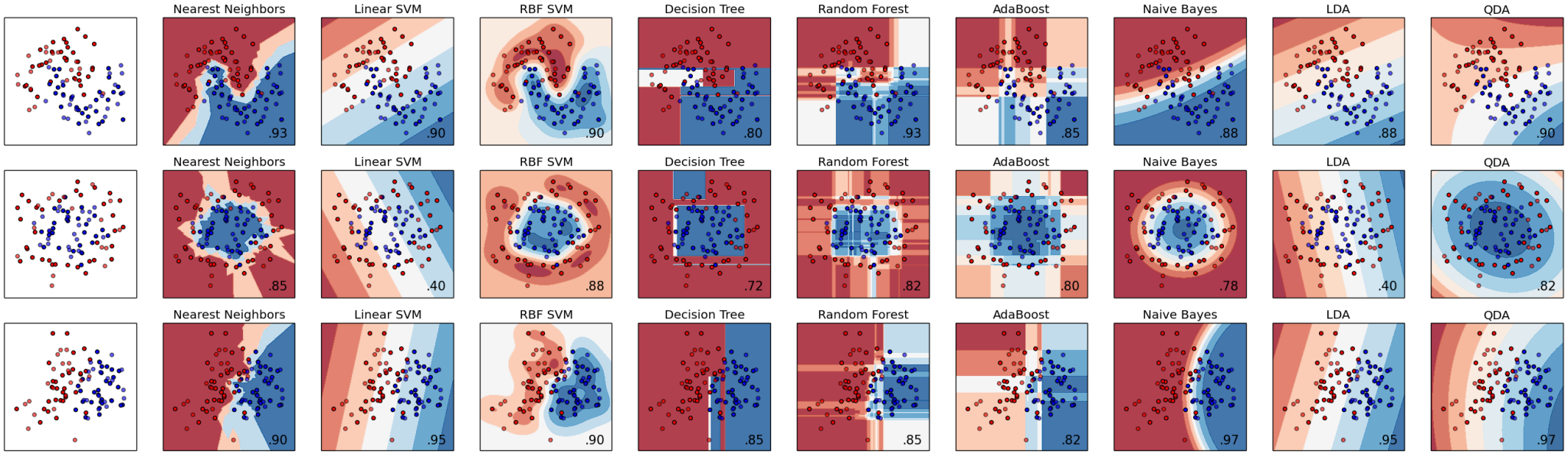
- Standardisation
- Outliers - low k-value is sensitive to outliers and a higher K-value is more resilient to outliers as it considers more voters to decide prediction
- Categorical values – create dummy variables
- Finding K (hyper-parameter) using cross-validation

Pros and Cons

Insensitive to outliers—accuracy can be affected from noise or irrelevant features
No assumptions about data—useful, for example, for nonlinear data
Easy to explain and to interpret
High accuracy (relatively)—it is pretty high but not competitive in comparison to better supervised learning models
Useful for regressions and classifications

Computationally expensive—because the algorithm stores all of the training data
High memory requirement
Prediction stage might be slow (with big N)
Sensitive to irrelevant features and the scale of the data

Algorithms for classification



https://scikit-learn.org/stable/supervised_learning.html

References

1. Classification algorithms - Data Science for Business, Foster Provost & Tom Fawcett (2013) O'REILLY
2. Clear Explanations Youtubers: Victor Lavrenko, edureka!, Luis Serrano
3. Classifications examples: elitedatascience.com, analyticsvidhya.com, listendata.com
4. Applied Predictive Modeling (2013) Kuhn, Max, Johnson, Kjell

https://github.com/BarbaraDiazE/CABANA_CHEMOINFORMATICS