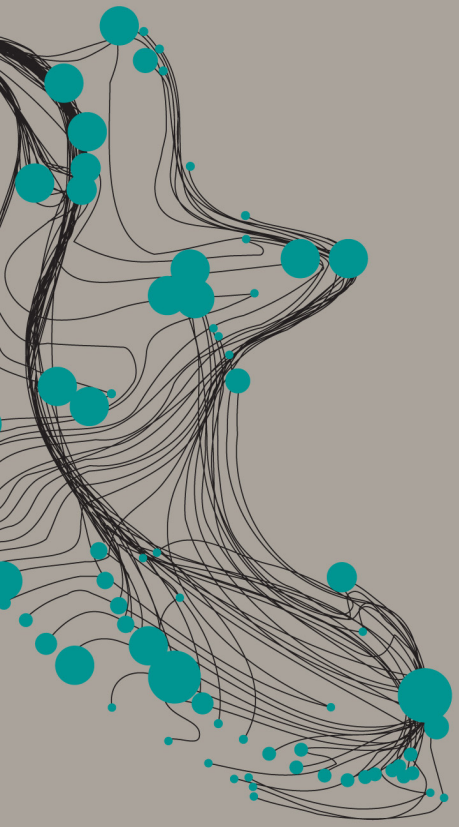
UNIVERSITY OF TWENTE.

Machine Learning Review

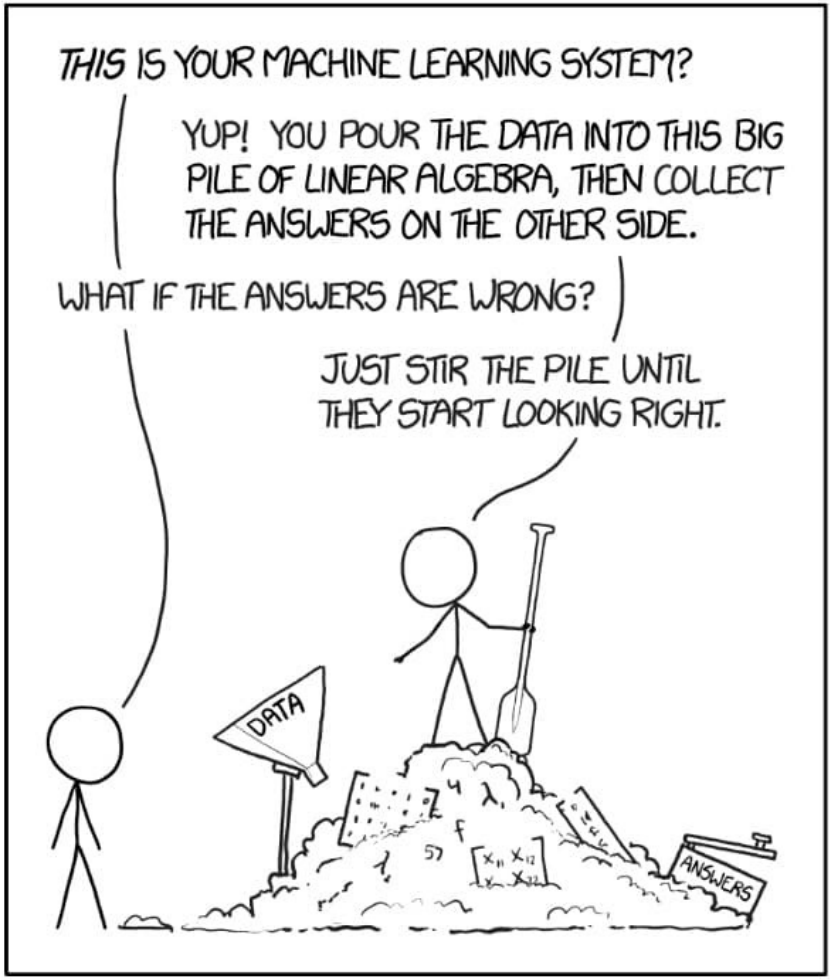
Mahdi KHODADADZADEH

October 2021



ITC FACULTY OF GEO-INFORMATION SCIENCE AND EARTH OBSERVATION

Machine Learning

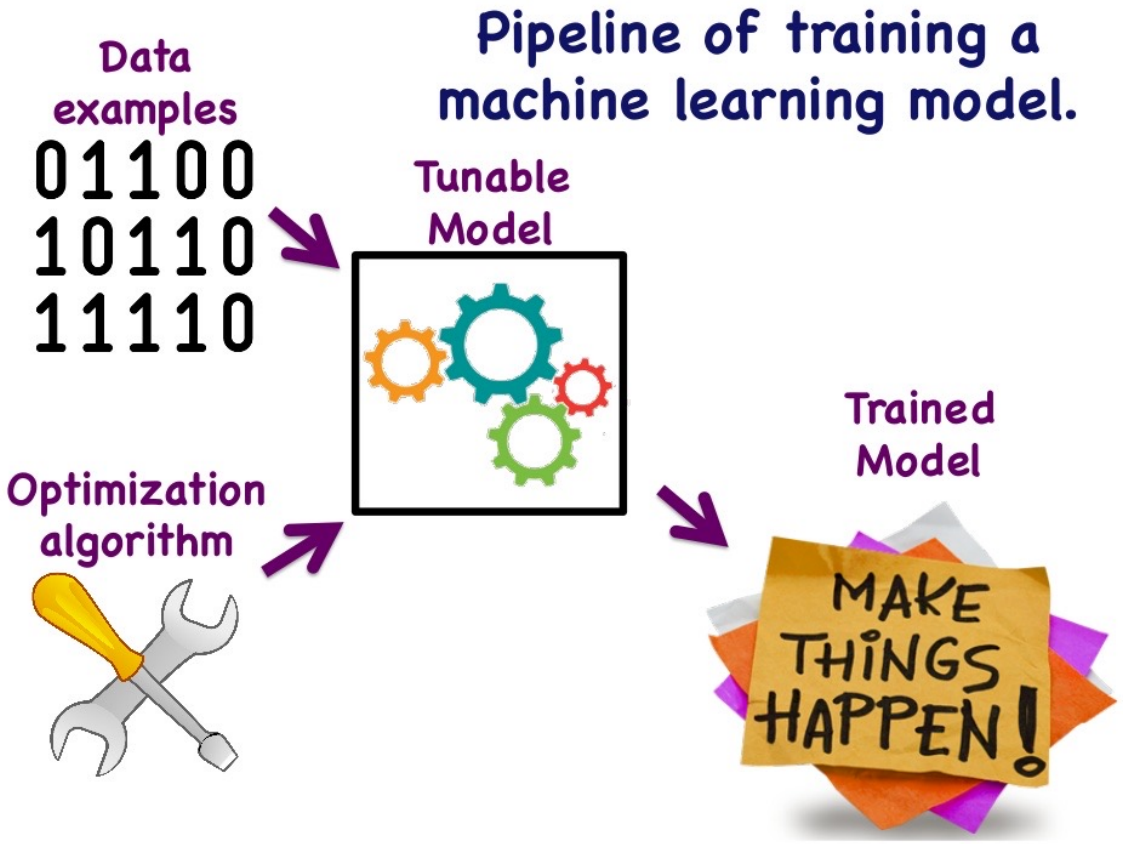


From: https://xkcd.com/

Machine Learning

- **When should we use Machine Learning**?
  - Problems where there are no human experts, so data cannot be labelled or categorized
  - Problems where there are human experts, but it is very hard (or impossible) to define rules
  - Problems where there are human experts and the rules can be defined, but where it is not cost effective to implement

Machine Learning



From : https://www.slideshare.net/kongqk/machine-learning-with-scikitlearn-77235584

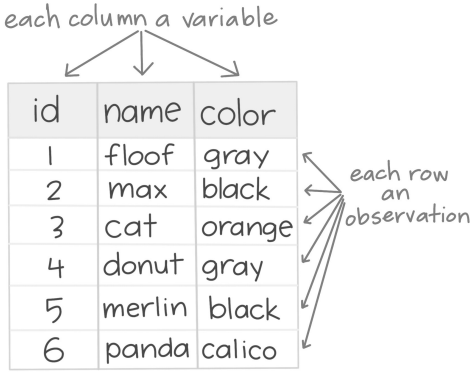
Machine Learning

"TIDY DATA is a standard way of mapping the meaning of a dataset to its structure."

–HADLEY WICKHAM

In tidy data:

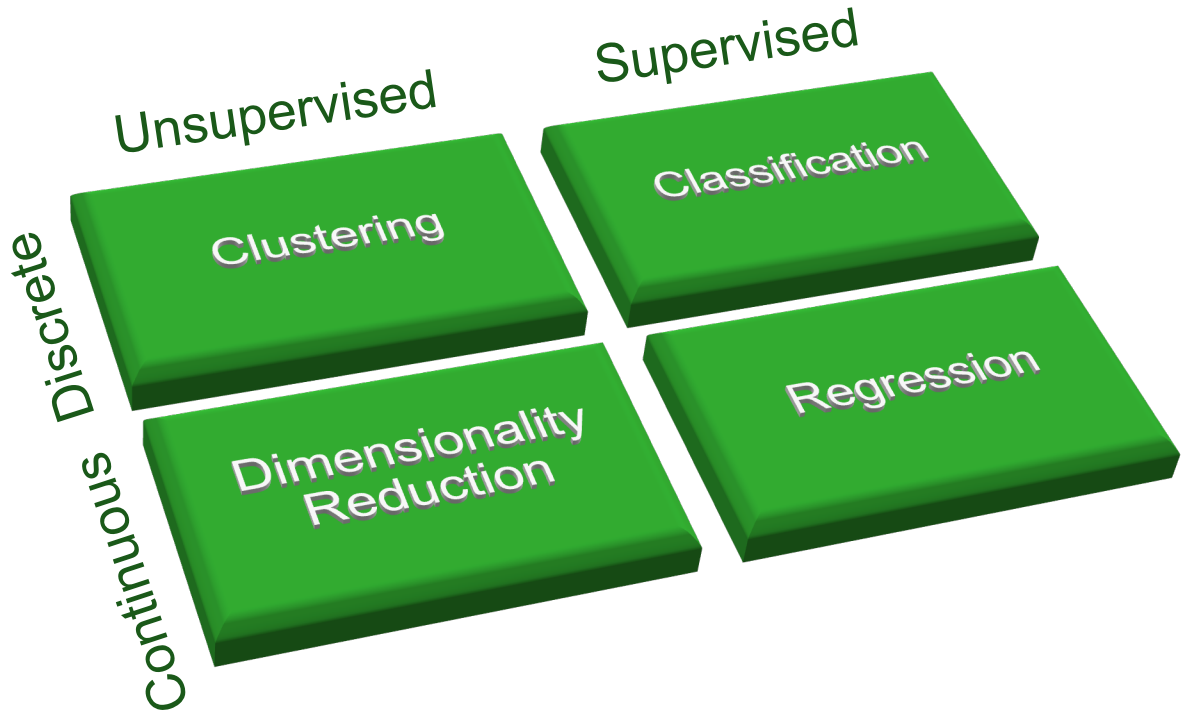
- each variable forms a column
- each observation forms a row
- each cell is a single measurement



Wickham, H. (2014). Tidy Data. Journal of Statistical Software *59* (10). DOI: 10.18637/jss.v059.i10

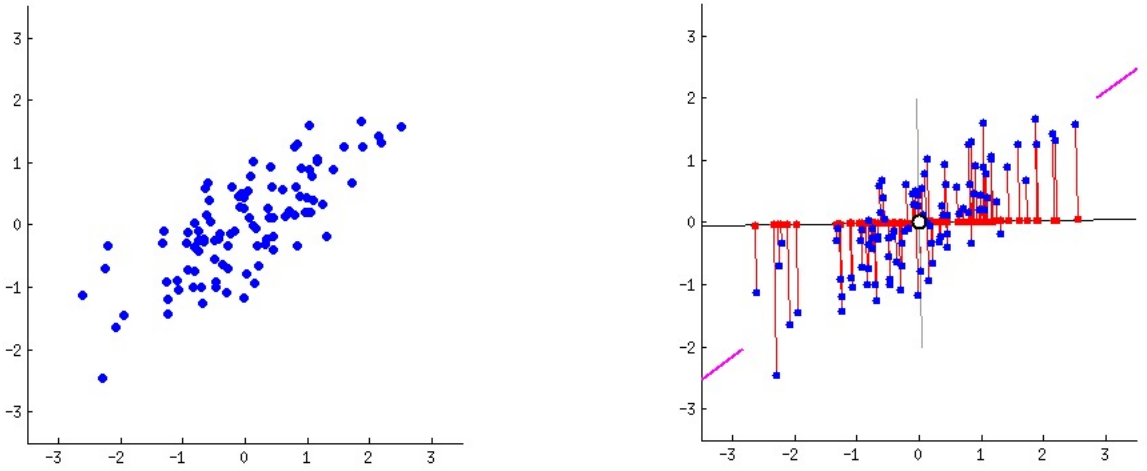
https://www.openscapes.org/blog/2020/10/12/tidy-data/

Machine Learning



Dimensionality Reduction

- **Principal component analysis (PCA)** converts a set of observations into a set of linearly uncorrelated variables, called principal components
- PCA represents data in a space that better describes the variation



From: https://stats.stackexchange.com/questions/2691/making-sense-of-principal-component-analysis-eigenvectors-eigenvalues

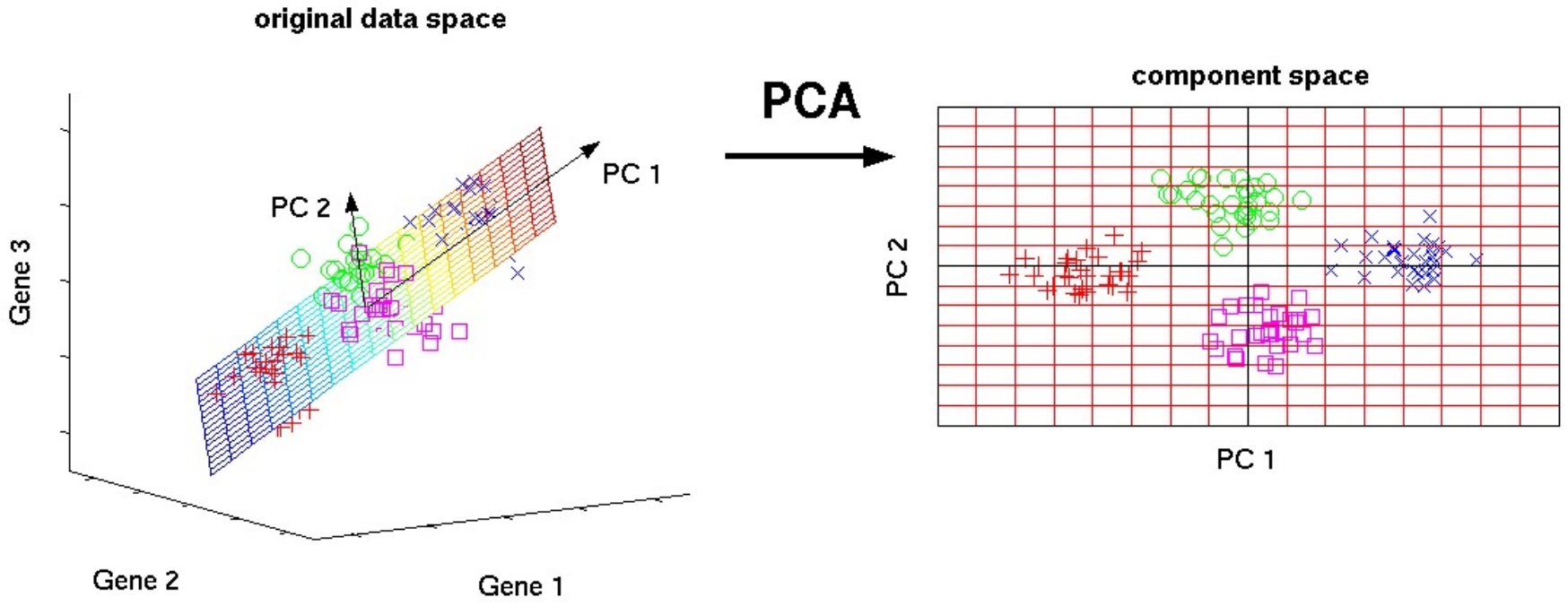
Dimensionality Reduction

Standardize the data

1. Obtain the Eigenvectors and Eigenvalues from the covariance matrix
2. Sort eigenvalues and pick the k eigenvectors that correspond to the k largest eigenvalues [k is the number of dimensions of the new feature subspace ($k \leq d$)]
3. Create the projection matrix W of the selected k eigenvectors.
4. Transform the original dataset X via W to obtain a k-dimensional feature subspace Y

$$Y_{nxk} = X_{nxd} W_{dxk}$$

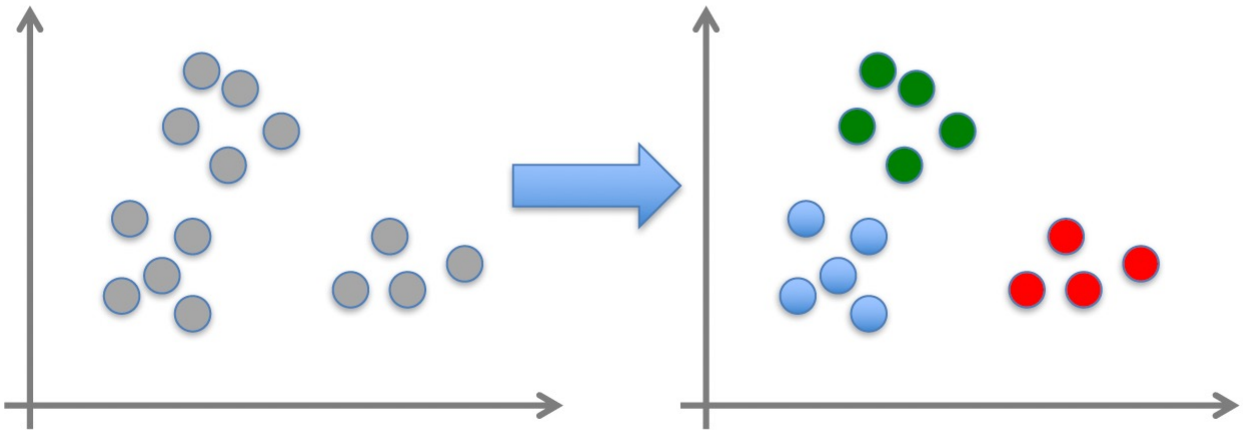
Dimensionality Reduction



From: http://www.nlpca.org/pca_principal_component_analysis.html

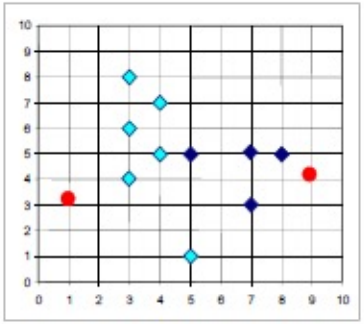
Clustering

- Identifying groups of elements that are similar among themselves but dissimilar to the elements in other groups.
- Learning using unlabeled data
- Given x1,x2,...,xn (without labels)
- Output hidden structure behind the x's

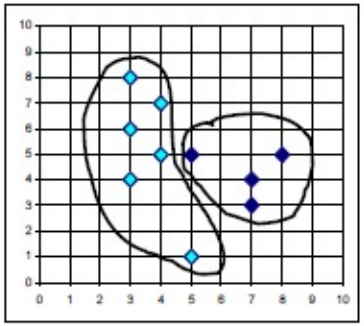


Clustering

1. Initialization: arbitrary initialization of the centers

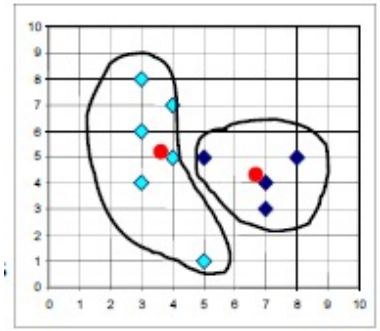


2. Data assignment. Each point is assigned to its closest cluster (center). Ties are broken by randomly assigning the point to one of the clusters. This yields a partitioning of the data.

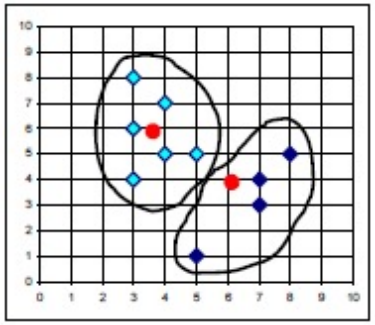


Clustering

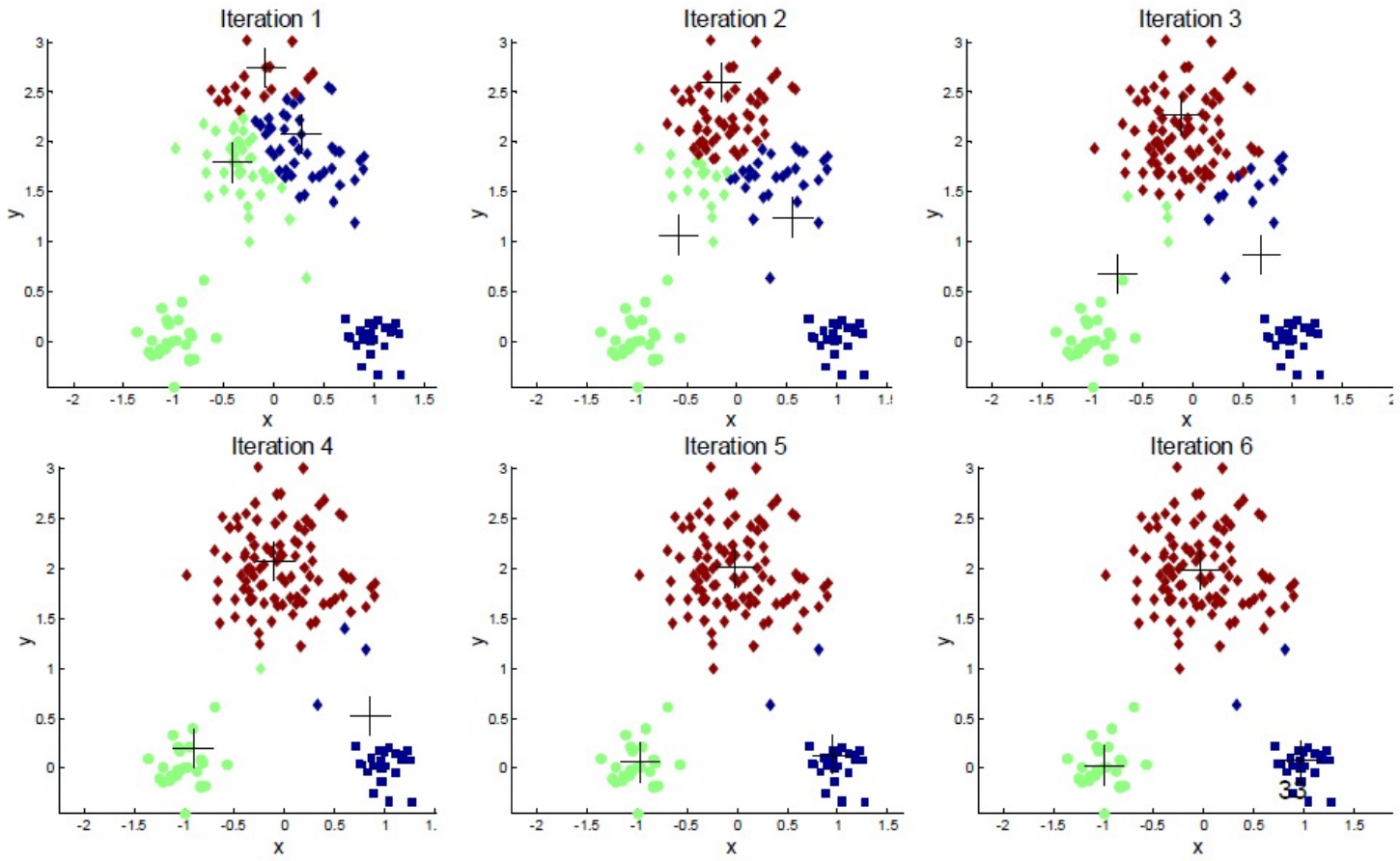
3. Relocating “centers”. Each cluster representative is moved to the center (arithmetic mean) of the points assigned to it



4. Repeat 2 and 3 until the centers do not longer change



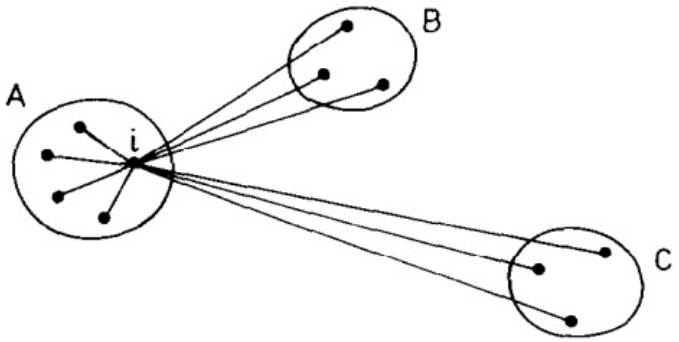
Clustering


Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
x
y

Optimal K value: silhouette

- Silhouette of a data instance is a measure of how closely it is matched to data within its cluster and how loosely it is matched to data of the neighbouring cluster
- A silhouette close to 1 implies the datum is in an appropriate cluster, while a silhouette close to −1 implies the datum is in the wrong cluster.
  - The Silhouette of data instances S(i) is calculated
  - Calculate the average S for increasing cluster numbers (e.g., 1 -10)
  - Finally, plot Average S – Number of Clusters

Optimal K value: silhouette



$a(i)$ = average dissimilarity of i to all other objects of A.

$d(i, C)$ = average dissimilarity of i to all objects of C.

$b(i) = \underset{C \neq A}{\text{minimum}}\ d(i, C)$.

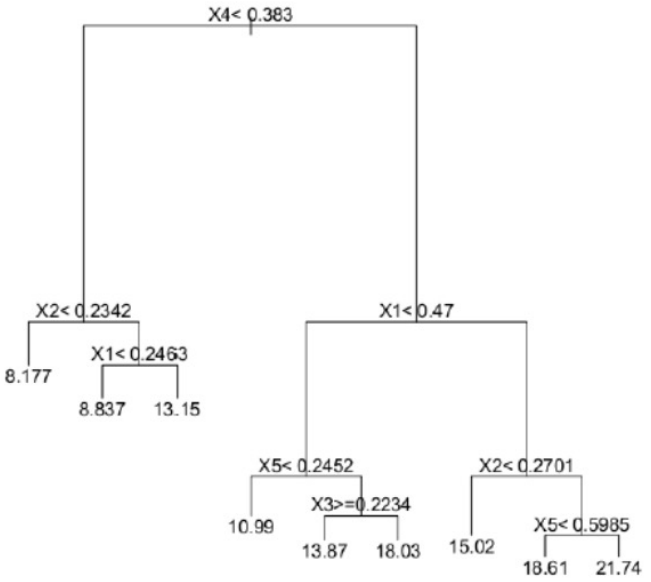
$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}.$$

$$s(i) = \begin{cases} 1 - a(i)/b(i) & \text{if } a(i) < b(i), \\ 0 & \text{if } a(i) = b(i), \\ b(i)/a(i) - 1 & \text{if } a(i) > b(i). \end{cases}$$

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20, 53-65.

Decision trees: CART

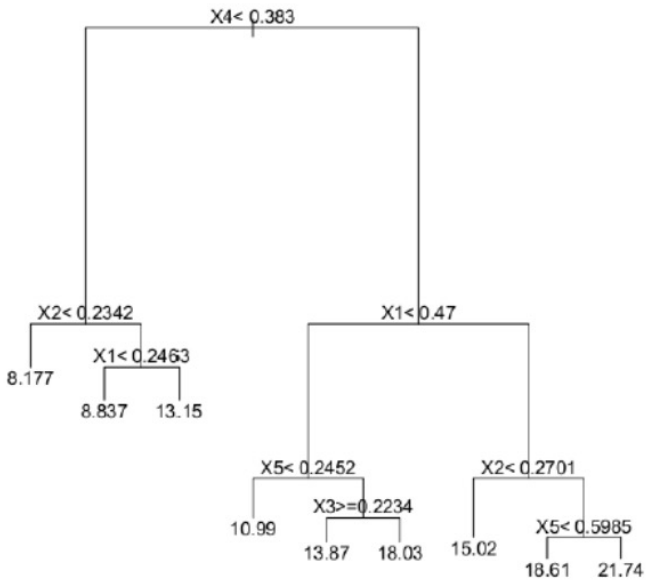
- Decision trees recursive partitioning of the data for classification and/or regression tasks
- CART: Classification and Regression Trees (Breiman et al., 1984)



- Start by creating the root node (all data)
- Root → 2 children → 4 grandchildren….
- "Grow" the tree until no further splits are possible :
  - all the data is the resulting node is homogeneous
  - it contains less elements than a (chosen) threshold

Decision trees: CART

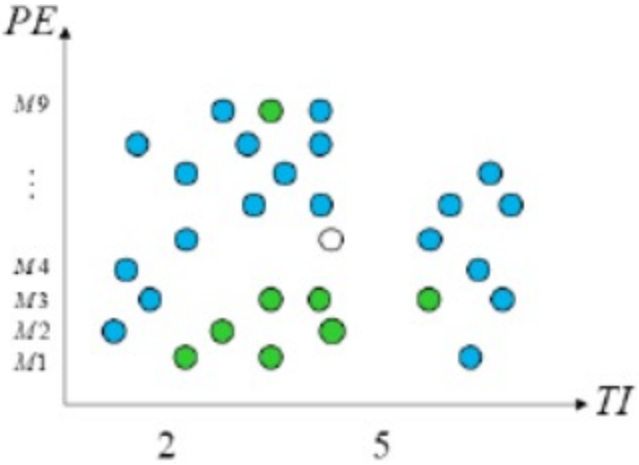
- An object goes left IF the chosen attribute meets some CONDITION, otherwise it goes right
  - Continuous data: X <= Condition
  - Nominal data: X belongs to set {A,B,C,D}

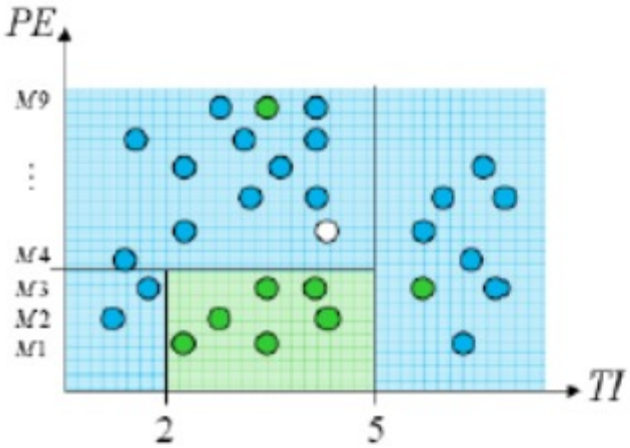


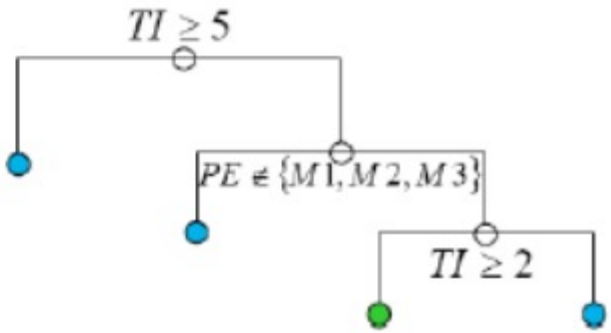
- The splitter and the split point are chosen by CART
  - Always binary splits
  - An attribute can be used multiple times

Decision Tree and Random Forest

TI	PE	Response
1.0	M2	good
2.0	M1	bad
...	...	...
4.5	M5	?



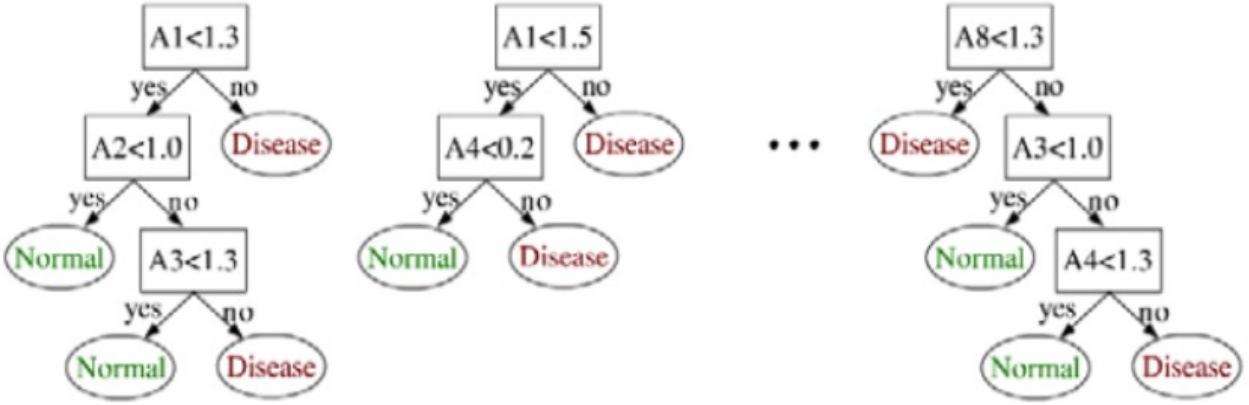




(Montillo, 2009)

Random forests

- Ensembles of trees
- Each tree is the result of applying the CART method to a random selection of attributes/features at each node
- And by using a random subset of the original input data (chosen with replacement, -- bootstrapping || Bagging = bootstrapping aggregation)
- Response variables are obtained by voting over the ensemble

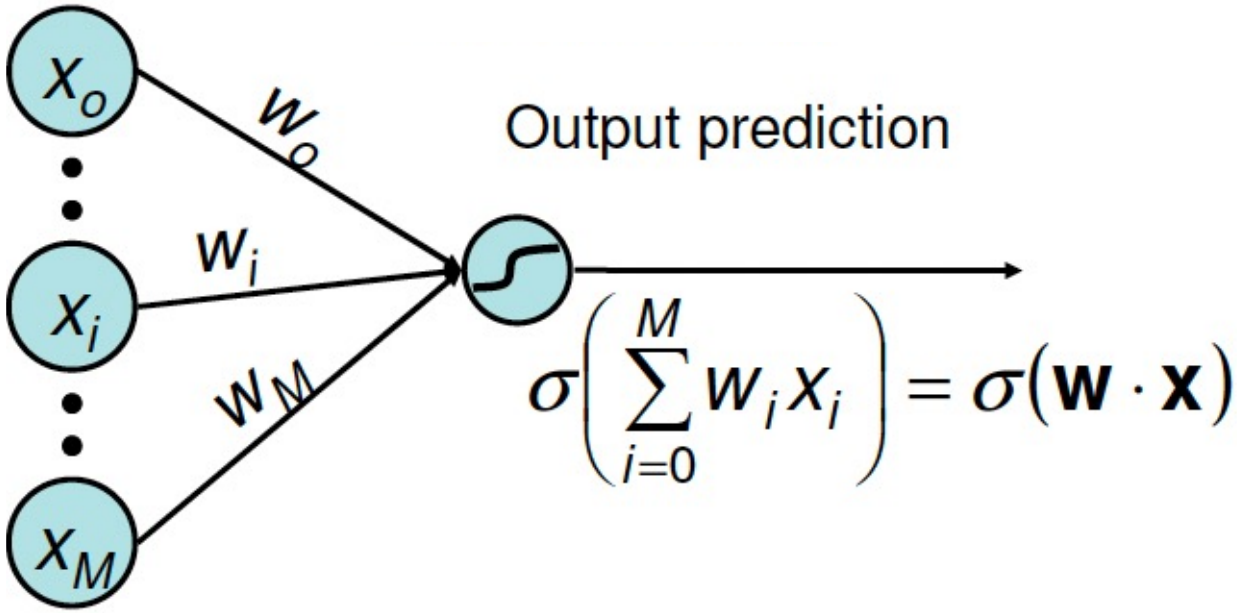


Random forests

- Input data: N training cases each with M variables
- n out of N samples are chosen with replacement (bagging).
- Rest of the samples to estimate the error of the tree (out of bag)
- m << M variables are used to determine the decision at a node of the tree
- Each tree is fully grown and not pruned

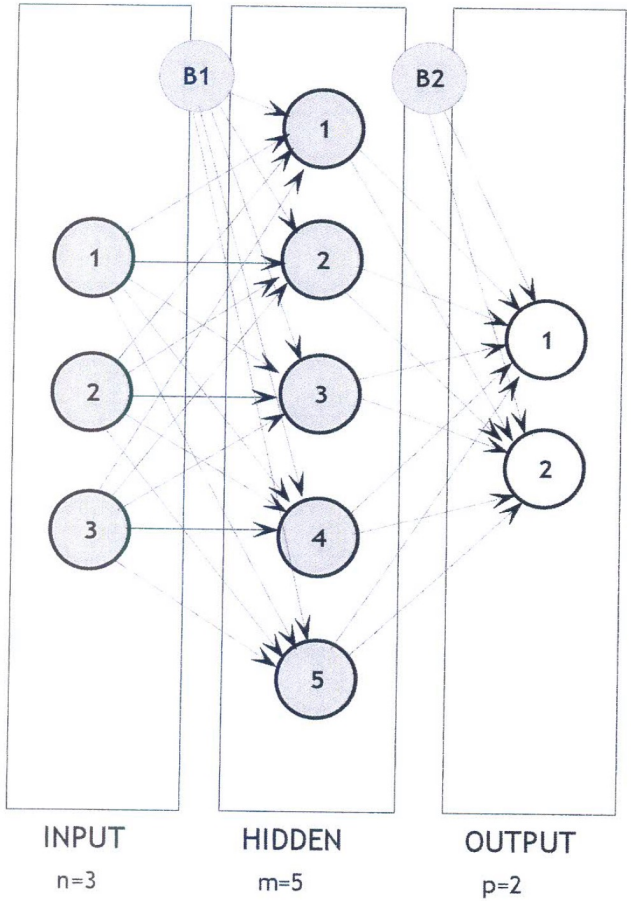
Artificial neural networks (ANNs)

- **ANNs** are machine learning models designed to imitate the human brain.
- Single-layer **Perceptron** → Input data are weighted, then added up and finally transformed using an activation function



Multi-layer perceptron (MLP)

- A special case of a feedforward neural network where every layer is a **fully connected layer**
- In **MLP** each node is connected to every node in the next layer
- Nodes in the hidden and output layers are connected to a bias node (feeding a constant value ~ constant in regression models)



How Many Layers and Nodes to Use?

- The input layer nodes take the values of the input features
- The number of neurons comprising **the input layer is equal to the number of features (columns)** in your data
  - Some configurations add one additional node for a bias term

- The **output layer has one node for each output**
  - A single node in the case of regression or binary classification
  - K nodes (with *softmax*) in the case of K-class classification

How Many Layers and Nodes to Use?

- For the hidden layers, use systematic experimentation to discover what works best for your specific dataset!
- In general, you cannot analytically calculate the number of layers and nodes

  → **hyperparameters optimization**

- Intuition and experience
- Read the literature
- First try other classifiers (e.g., Random Forest)!

Hyperparameter Optimization (HPO)

- Grid search
- Random search

Performance Evaluation: Regression

- Each residual contributes to the total error
- How you want your model to treat outliers, or extreme values, in your data?

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

$$MSE = \frac{1}{n}\Sigma\left(y - \widehat{y}\right)^2$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(Predicted_i - Actual_i)^2}{N}}$$

- **R-Squared** defines the degree to which the variance in the dependent variable (target or response) can be explained by the independent variable (features or predictors)
- Tells how good a model is performing

$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \overline{y})^2}$$

Performance Evaluation: Binary Classification

- **Accuracy: (TP+TN)/total** → Overall, how often is the classifier correct?
- **True Positive Rate (Sensitivity or Recall): TP/Actual Yes** → When it's actually yes, how often does it predict yes?
- **True Negative Rate (Specificity): TN/Actual NO** → When it's actually no, how often does it predict no?
- **Precision: TP/predicted yes** → When it predicts yes, how often is it correct?

$$F_1\text{−score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}$$

Performance Evaluation: Multi-class Classification

- **Confusion matrix** for multi-class classification

	Predicted class 1		…	Predicted class n
Actual class 1	C_{11}	C_{12}	…	C_{1n}
Actual class 2	C_{21}			
	…			
Actual class n	C_{n1}			C_{nn}

Overall Accuracy:
$(C_{11} + C_{22} + \ldots + C_{nn})$/total

Class "i" Accuracy:
(C_{ii})/Actual class "i"

Average (Class) Accuracy:
Average of all the class accuracies

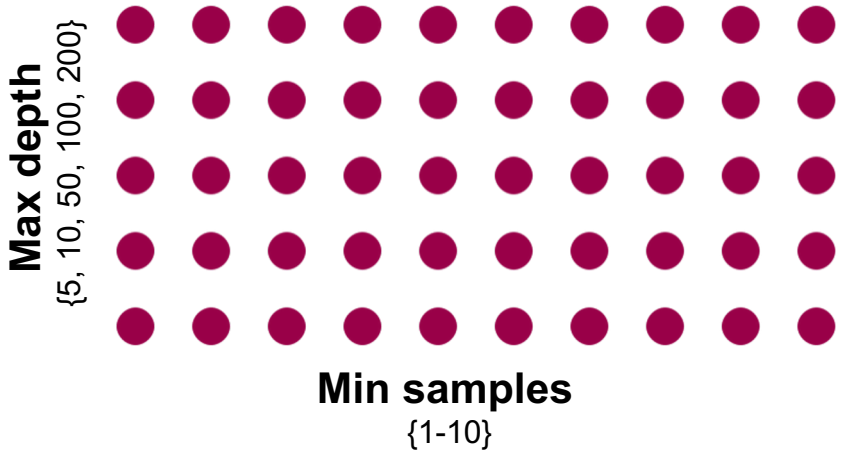
Hyperparameter Optimization 1

- Also called **metaparameter optimization** or **hyperparameter tuning**
- Any system that chooses hyperparameters automatically
- **HPO** *aims to improve the performance of a model by choosing the right set of hyperparameters.*
- What it does:
  - *Select a set of parameters (hyperparameters)*
  - *Train a model*
  - *Evaluate the performance of the model (validation set)*
  - *Move on to the next set*
  - *Keep the hyperparameters that improves the performance the most*
- **It is computationally expensive!**

Hyperparameter Optimization 2: grid search

- A traditional technique that forces all combinations
- **Search space** = possible combinations of parameters
  - Define some grid of parameters you want to try
  - Try all the parameter values in the grid
  - Then choose the setting with the best result

DT: max depth,
min_samples split



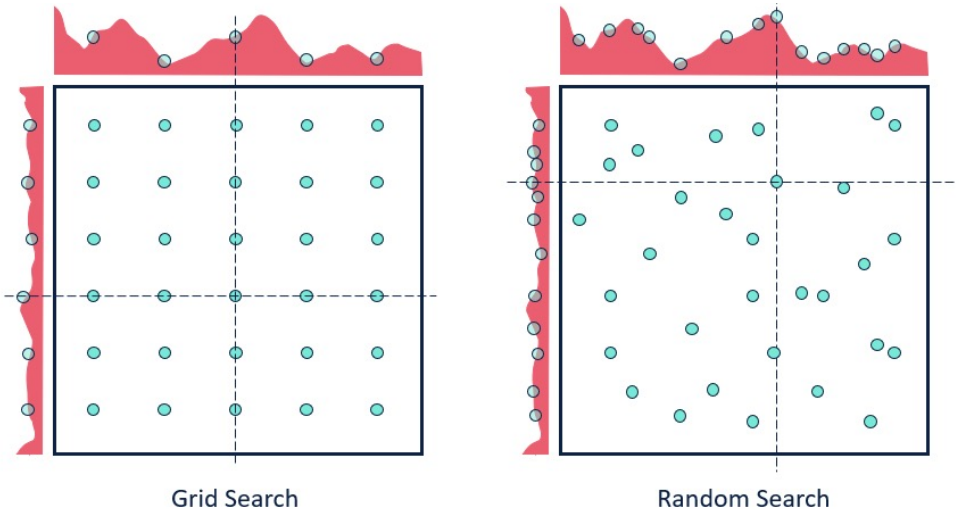
Hyperparameter Optimization 3: grid search

- Does not scale well…
- No warrantee that you explore the right search space
- Strategy → a two stage grid search
  - **large scale, then regional focus**
    - Max depth { 10, 50, **100**, 200} / Min samples {1, **5**, 10}
      - *Max depth { **80**, 100, 120} / Min samples {4, 5, **6**}*

- Take advantage of parallelism
  - Run all the different parameter settings independently on different servers in a cluster

Hyperparameter Optimization 4: random search

- This is just grid search, but with randomly chosen points instead of points on a grid.
- Randomly sample the search space
- Not necessarily going to get anywhere near the optimal parameters (may miss the best combination)
- Much faster!



From: https://community.alteryx.com/t5/Data-Science/Hyperparameter-Tuning-Black-Magic/ba-p/449289

Hyperparameter Optimization 5: Bayesian Optimization

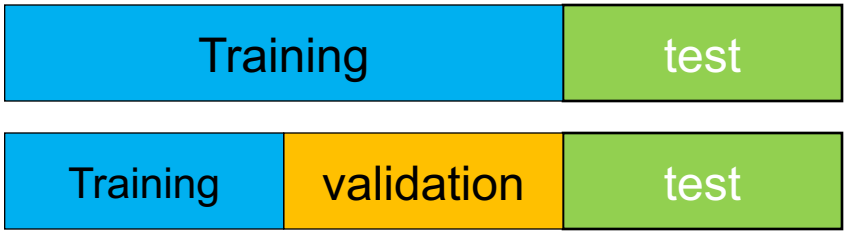
- Learns a statistical model of the function from hyperparameter values to the loss function
- Then choose parameters to minimize the loss under this model
- Main benefit: choose the hyperparameters to test not at random, but in a way that gives the most information about the model
- Faster than grid search
- Not as simple-to-implement as grid search!

Model validation

- Holdout method
- K-fold cross-validation
- Backtesting

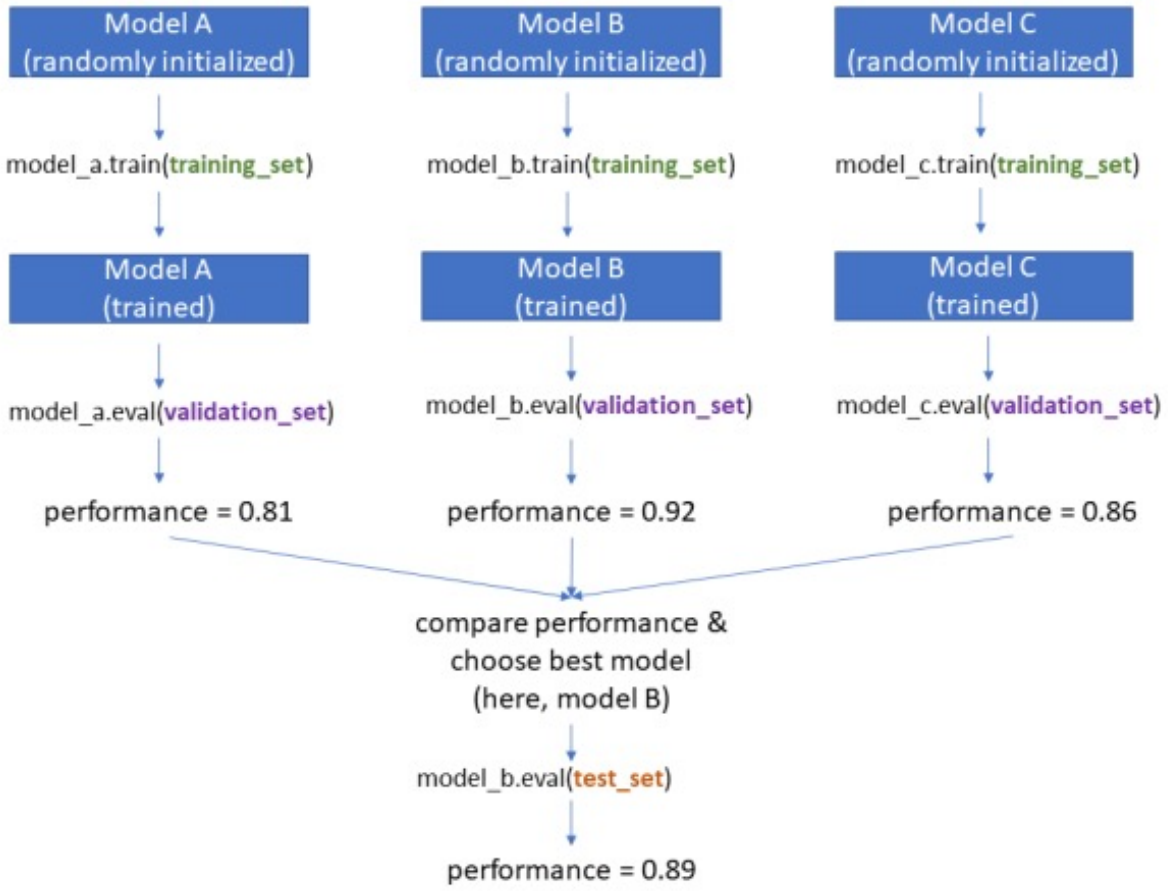
Model validation: holdout

- **Model Validation:** Any process by which a model is verified against additional data not used in the process of generating the model.



- The validation set is a separate section of your dataset that you will use during training to get a sense of how well your model is doing (evaluation of a model fit on the training dataset while tuning model hyperparameters)
- The split percentage depends on the total number of samples in your data and the actual model you are training
- Common ratios train/val/test:
- 70/15/15 or 80/10/10 or 60/20/20

Model validation: holdout



From: https://glassboxmedicine.com/2019/09/15/best-use-of-train-val-test-splits-with-tips-for-medical-data/

Model validation: k-fold cross-validation

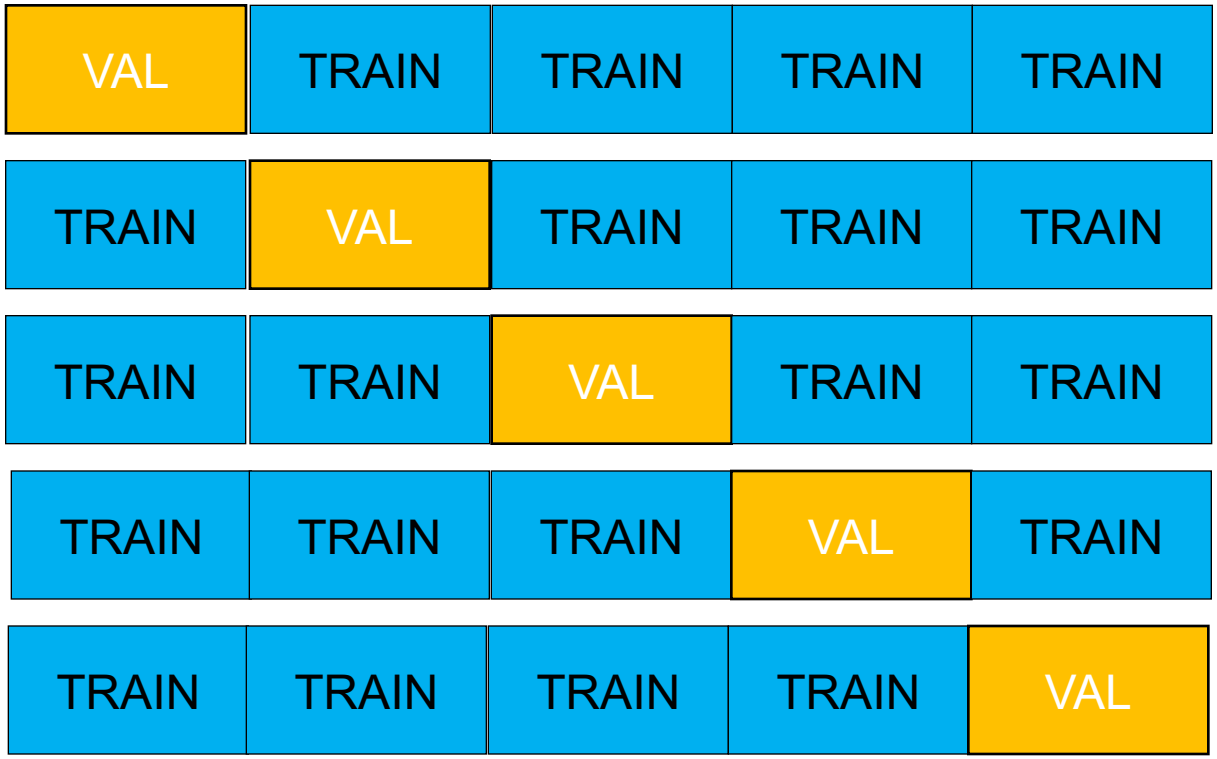
- Holdout → Lose a good amount of data

- Cross Validation: A type of model validation where multiple subsets are created and verified against each-other, usually in an iterative approach requiring the generation of a number of separate models equivalent to the number of subsets created.
- Cross validation avoids over fitting.
- It allows an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters.

- Basically, you use your training set to generate multiple splits of the Train and Validation sets.

Model validation: k-fold cross-validation

- Extrapolate from 2 to k → k-fold cross validation



Equal subsets

K=5 → 5 iterations

Model validation: backtesting

- Holdout method & *k*-fold cross-validation do not work for time series forecasting….

They assume that there is no relationship between the observations, that each observation is independent.

- **Backtesting:** the evaluation of models on historical data
  - **Holdout** that respect temporal order of observations.
  - **Multiple Train-Test splits** that respect temporal order of observations.
    - TimeSeriesSplit (Scikit)

Model validation: the modelling cycle

Input data → then, Train/Test

HPO

(CV +Performance Evaluation)

Model Validation

Apply different methods

Compare performance of methods

Optimal model → **PREDICT**

Python

sklearn.model_selection.RandomizedSearchCV

class sklearn.model_selection.RandomizedSearchCV(*estimator*, *param_distributions*, ***, *n_iter=10*, *scoring=None*, *n_jobs=None*, *refit=True*, *cv=None*, *verbose=0*, *pre_dispatch='2*n_jobs'*, *random_state=None*, *error_score=nan*, *return_train_score=False*) [source]

Randomized search on hyper parameters.

RandomizedSearchCV implements a "fit" and a "score" method. It also implements "score_samples", "predi
"decision_function", "transform" and "inverse_transform" if they are implemented in the estimator used.

The parameters of the estimator used to apply these methods are optimized by cross-validated search ove
settings.

In contrast to GridSearchCV, not all parameter values are tried out, but rather a fixed number of parameter
from the specified distributions. The number of parameter settings that are tried is given by n_iter.

If all parameters are presented as a list, sampling without replacement is performed. If at least one paramet
distribution, sampling with replacement is used. It is highly recommended to use continuous distributions fo
parameters.

Read more in the User Guide.

New in version 0.14.

Parameters:

estimator : *estimator object*
A object of that type is instantiated for each grid point. This is assumed to implement th
estimator interface. Either estimator needs to provide a `score` function, or `scoring` mu

param_distributions : *dict or list of dicts*
Dictionary with parameters names (`str`) as keys and distributions or lists of parameters
must provide a `rvs` method for sampling (such as those from scipy.stats.distributions).
sampled uniformly. If a list of dicts is given, first a dict is sampled uniformly, and then a
sampled using that dict as above.

n_iter : *int, default=10*
Number of parameter settings that are sampled. n_iter trades off runtime vs quality of t

scoring : *str, callable, list, tuple or dict, default=None*
Strategy to evaluate the performance of the cross-validated model on the test set

sklearn.model_selection.GridSearchCV

class sklearn.model_selection.GridSearchCV(*estimator*, *param_grid*, ***, *scoring=None*, *n_jobs=None*, *refit=True*, *cv=None*, *verbose=0*, *pre_dispatch='2*n_jobs'*, *error_score=nan*, *return_train_score=False*) [source]

Exhaustive search over specified parameter values for an estimator.

Important members are fit, predict.

GridSearchCV implements a "fit" and a "score" method. It also implements "score_samples", "predict", "predict_proba", "decision_function", "transform" and "inverse_transform" if they are implemented in the estimator used.

The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

Read more in the User Guide.

Parameters:

estimator : *estimator object*
This is assumed to implement the scikit-learn estimator interface. Either estimator needs to provide a `score` function, or `scoring` must be passed.

param_grid : *dict or list of dictionaries*
Dictionary with parameters names (`str`) as keys and lists of parameter settings to try as values, or a list of such dictionaries, in which case the grids spanned by each dictionary in the list are explored. This enables searching over any sequence of parameter settings.

scoring : *str, callable, list, tuple or dict, default=None*
Strategy to evaluate the performance of the cross-validated model on the test set.

If `scoring` represents a single score, one can use:

- a single string (see The scoring parameter: defining model evaluation rules);
- a callable (see Defining your scoring strategy from metric functions) that returns a single value.

If `scoring` represents multiple scores, one can use:

- a list or tuple of unique strings;
- a callable returning a dictionary where the keys are the metric names and the values are the metric scores;
- a dictionary with metric names as keys and callables a values.