A **Aprendizagem Simbólica** (AS) refere-se ao facto de todos os passos se basearem em **representações simbólicas de leitura humana** do problema que utilizam a lógica e a procura para resolver o problema.

- A principal vantagem da AS é que o processo de raciocínio pode **ser facilmente explicado** - num programa de AS é facil perceber porque é que se chega a uma determinada conclusão e quais foram os passos do raciocínio.
- Uma das desvantagens principais da AS é que, para o processo de aprendizagem – as **regras e o conhecimento precisam ser codificados à mão**, o que é um problema difícil.
- Até agora, a AS está muito confinada ao mundo acadêmico e laboratórios universitários com pouco investimento dos gigantes da indústria.

- Uma das desvantagens principais da Aprendizagem não simbolica (AnS) é que é **difícil compreender** como é que o sistema chegou a uma conclusão. Isto é particularmente importante quando aplicado a aplicações críticas, tais como condução autonoma de automóveis, diagnóstico médico, entre outras.
- Em sistemas não simbólicos, como aplicações alimentados por DL, não são aceitáveis **decisões de alto risco**

Aprendizagem supervisionada
(*Supervised Learning*)

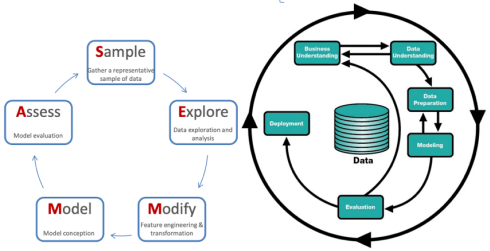
- Classificação (Classification)
  - Classificação de imagem
  - Fidelização de clientes
  - Diagnóstico
  - Deteção de fraude
- Regressão (Regression)
  - Crescimento populacional
  - Previsão de mercados
  - Previsão meteorológica
  - Esperança média de vida

Aprendizagem não supervisionada
(*Unsupervised Learning*)

- Redução (*Reduction*)
  - Visualização (Big Data)
  - Compreensão de significados
  - Descoberta de Estruturas
  - Seleção de atributos
- Segmentação (*Clustering*)
  - Segmentação de clientes
  - Marketing
  - Sistemas de recomendação

Aprendizagem por reforço
(*Reinforcement Learning*)

- Aquisição de aptidões
- Navegação de robôs
- Decisões em tempo real
- Tarefas de aprendizagem
- Jogos com IA



A set of basic data preparation techniques can be used:

- Union/intersection of columns;
- Concatenation
- Sorters
- Filters (column, row, nominal, rule-based, ...)
- Basic aggregations (counts, unique, mean/sum, ...)

- Feature scaling
- Outlier detection
- Feature selection
- Missing Values treatment
- Nominal value discretization
- Binning
- Feature Engineering

1. Normalizing the range of the independent features

   - Standardization (or Z-score Normalization): Rescaling the distribution of values so that the mean of observed values is 0 and the standard deviation is 1. Assumes observations fit a Gaussian distribution with a well-behaved mean and standard deviation, which may not always be the case.
   - Normalization: Rescaling data so that all values fall within the range of 0 and 1, for example.

Feature Selection (or dimensionality reduction):

Rationale: which features should we use to create a predictive model? Select a sub-set of the most important features to reduce dimensionality.

The removal of unimportant features:

- May affect significantly the performance of a model
- Reduces overfitting (less opportunity to make decisions based on noise)
- Improves accuracy
- Helps reducing the complexity of a model (reduces training time)

Feature Selection (or dimensionality reduction):

- Remove a feature if the percentage of **missing values** is higher than a threshold;
- Use the chi-square test to measure the **degree of dependency** between a feature and the **target class**;
- Remove feature if low standard deviation;
- Remove feature if data are highly skewed;
- Remove features that are highly correlated between each other.

5. Nominal value discretization:

Rationale: categorical data often called nominal data, are variables that contain label values rather than numeric ones. Several methods may be applied:

- One-Hot Encoding
- Label Encoding
- Binary Encoding

Nominal value discretization:

Movie	Genre
Jumanji	Adventure
American Pie	Comedy
Braveheart	Drama
...	...

Label Encoded

Movie	Genre	Category
Jumanji	Adventure	0
American Pie	Comedy	1
Braveheart	Drama	2
...	...	

One-Hot Encoded

Movie	Adventure	Comedy	Drama
Jumanji	1	0	0
American Pie	0	1	0
Braveheart	0	0	1
...	...		

Binning, i.e., group numeric data into intervals - called bins:

Rationale: make the model more robust and prevent overfitting. However, it penalizes the model's performance since every time you bin something, you sacrifice information.

Discrete dependent variable: **classification** problem

Logistic regression: uses regression models for **binary classification** by interpreting the **model output** in order to extract a **class**

Logistic regression can be applied to cases with **more than two classes**

In this case, the strategy is **to train a "binary" model for each class separately** (considering the others as a single class)

The parameter **α** is called the **learning rate** and controls the **"speed"** of updating the parameters

Lower α values guarantee convergence but it may be slower

Higher α values can lead to **faster convergence**, but carry risks of divergence

Solutions for Overfitting:

Reduce the number of attributes (coefficients) used

Select attributes "**manually**" by knowledge of the problem

Attribute selection algorithms

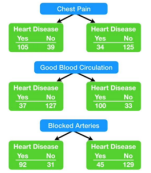
Underfitting: insuficient complexity

"Adequate" complexity

Overfitting: excessive complexity

Classification Trees

Chest Pain	Good Blood Circulation	Blocked Arteries	Heart Disease
No	No	No	No
Yes	Yes	Yes	Yes
Yes	Yes	No	No
Yes	No	???	Yes
etc...	etc...	etc...	etc...



$G_{ChestPain} = 0.364$

$G_{BloodCirculation} = 0.360$

$G_{BlockedArteries} = 0.381$

Method:
1 – calculate the Geni Impurity scores
2 – If node has lowest score do not separate any more and becomes leaf
3 – If separating becomes an improvement than choose the separation with the lowest impurity score

Classification Trees – continuous attributes

Gini = 0.3

Method (**continuous attributes**):
1 – sort the attribute lowest to highest
2 – calculate the average value for all adjacent attributes
3 – calculate the impurity values for each average value
4 – if separating becomes an improvement than choose the separation with the lowest impurity score

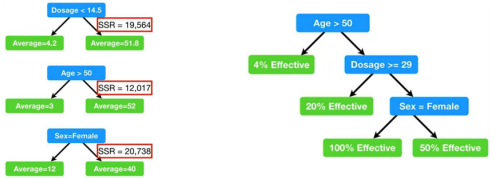
Method

1 - For each possible threshold calculate the average of the left and right samples and calculate the sum of the square error for each sample

2 – select the threshold with the minimum sum of squared errors for a branch

3 – when number of samples less then predefined value (e.g. 20) then it is a leaf with value equal to the average of samples

Regression Trees – multiple attributes



Method
1 – calculate the minimum of the sum of squared errors for each atribute
2 – select the attribute and threshold with the minimum sum of squared errors for a branch
3 – when number of samples less then predefined value (e.g. 20) then it is a leaf with value equal to the average of samples

- **min_sample_leaf** on the other hand is basically the minimum no. of sample required to be a leaf

A decision tree will always overfit the training data if we allow it to grow to its max depth

Pre-prunnig (early stopping):
- **min_sample_split** is the minimum no. of sample required for a split.

Post-prunnig (after perfect training):
- Assign a maximum depth to a tree

Strengths:
- **Simple configuration** (doesn't have too many configuration parameters);
- Compared to other algorithms decision trees requires **less effort for data preparation** during pre-processing.
- A decision tree does **not require normalization** of data.
- A decision tree does **not require scaling** of data as well.
- **Missing values in the data also do NOT affect** the process of building a decision tree to any considerable extent.
- A Decision tree model is **very intuitive and easy to explain** to technical teams as well as stakeholders.

Weaknesses:
- **Inadequate** for problems characterized by **many interactions between attributes**;
- Does not avoid **replicas** of subtrees;
- A **small change in the data** can cause a **large change in the structure** of the decision tree causing instability.
- For a Decision tree sometimes calculation can go far **more complex** compared to other algorithms.
- Decision tree often involves **higher time to train** the model.

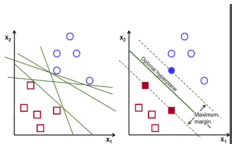
Support Vector Machine is a **supervised Machine Learning algorithm** that can be used for both classification (mostly) or regression problems.

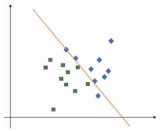
Usually a learning algorithm tries to **learn the most common characteristics** (what differentiates one class from another) of a class and the classification is based on those representative characteristics learnt (classification is based on differences between classes). The SVM works in the other way around. It finds the **most similar examples between classes**. Those will be the support vectors.

Works well for **classifying higher-dimensional data** (datasets with lots of features)

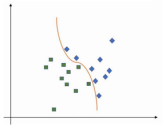
Finds higher-dimensional support vectors which "divides" the data

Applies kernels to represent data in higher-dimensional spaces to find hyperplanes that might not be apparent in lower dimensions

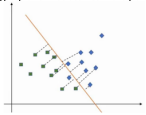




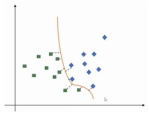
Low Regularization (C) - the optimizer to look for a **larger-margin** separating hyperplane, even if that hyperplane misclassifies more points.



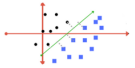
High Regularization (C) - the optimization will choose a **smaller-margin** hyperplane if that hyperplane does a better job of getting all the training points classified correctly



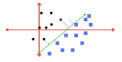
Low Gamma - **points far away** from plausible hyperplane are considered in calculation for the hyperplane



High Gamma - **only points close** to plausible line are considered in calculation



Good margin – equidistant as far as possible from both sides



Bad margin- very close to blue class

The **multiclass problem is broken down to multiple binary classification cases**, which is also called **one-vs-one**. In scikit-learn one-vs-one is not default and needs to be selected explicitly. **One-vs-rest** is set as default. It basically divides the data points in class x and rest. Consecutively a certain class is distinguished from all other classes.

Support Vector Regression

Strengths:

Although Support Vector Regression is used rarely it carries certain advantages:
- It is **robust to outliers**.
- Decision model can be **easily updated**.
- It has excellent **generalization capability**, with high prediction accuracy.
- Its implementation is easy.

Weaknesses:

Some of the drawbacks faced by Support Vector Machines while handling regression problems are:
- They are **not suitable for large datasets**.
- In cases where the number of features for each data point exceeds the number of training data samples, the SVM will underperform.
- The Decision model **does not perform very well when the data set has more noise** i.e. target classes are overlapping.

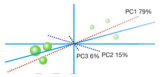
Support Vector Machines

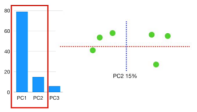
Strengths:
- Effective on datasets with **multiple features**, like financial or medical data.
- Effective in cases where **number of features is greater than the number of data points**.
- Uses a **subset of training points** in the decision function called support vectors which makes it memory efficient.
- Different **kernel functions** can be specified for the decision function. You can use common kernels, but it's also possible to specify custom kernels.

Weaknesses:
- If the number of features is a lot bigger than the number of data points, avoiding over-fitting when **choosing kernel functions and regularization term is crucial**.
- SVMs don't directly provide probability estimates. Those are calculated using an expensive n-fold cross-validation.
- Works **best on small sample sets** because of its high training time.

Principal Component Analysis (PCA) uses the projection of the original data into the principal components. The principal components are orthogonal vectors that describe the maximum amount of residual variation.

1 - **Standardize** the data
2 - Construct a **correlation matrix**
3 - Calculate the **eigenvectors/unit vectors and eigenvalues**. Eigenvalues are scalars by which we multiply the eigenvector of the covariance matrix.
4 - Sort the eigenvectors in highest to lowest order and **calculate the percentage of variation** that each PC accounts for.
5 - **Select the number of components**, (the so-called elbow method can be used). Plot a graph of the cumulative sum of the explained variance and then select the number of components that explains the desired ratio of information (usually 80% or 95%).

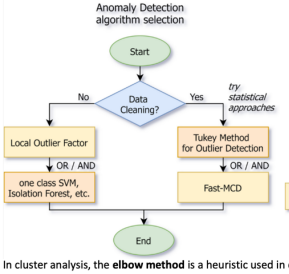


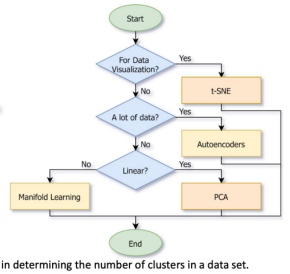


PCA versions:
Incremental PCA - for online learning or when data doesn't fit in memory
Randomized PCA - stochastic algorithm that allows to quickly estimate the first N components
Kernel PCA - kernel trick allows performing complex nonlinear projections

Dimensionality reduction – Autoencoders

Autoencoder is a artificial neural network that tries to output values that are as similar as possible to the inputs when the network structure implies *a bottleneck* - a layer where the number of neurons is much fewer than in the input layer.





In cluster analysis, the **elbow method** is a heuristic used in determining the number of clusters in a data set. The method consists of plotting the explained variation as a function of the number of clusters and picking the elbow of the curve as the number of clusters to use. The same method can be used to choose the number of parameters in other data-driven models, such as the number of principal components to describe a data set.

All clustering algorithms require data preprocessing (e.g. dimensionality reduction) and standardization.

K-Means - Algorithm based on the **centroid concept**. Centroid is a geometric center of a cluster (mean of coordinates of all cluster points).

Strengths:
- **Simple** and intuitive;
- Scales to **large datasets**;
- As a result, we also have **centroids** that can be used as standard cluster representatives

Weaknesses:
- Knowledge about the **number of clusters** is necessary and must be specified as a parameter;
- Does not cope well with a very large number of features;
- Separates only convex and homogeneous clusters well;
- Can result in poor local solutions, so it needs to be run several times.

Hierarchical clustering (also Hierarchical Cluster Analysis (HCA) or Agglomerative Clustering) is a family of clustering algorithms that build a hierarchy of clusters during the analysis:
- Start with points as individual clusters.
- At each step, merge the closest pair of clusters until only one cluster (or K clusters left).

Strengths :
- Simple and intuitive;
- Works well when data has a hierarchical structure;
- Knowledge about the number of clusters is not necessary.

Weaknesses :
- Requires additional analysis to choose the resulting number of clusters;
- Separates only convex and homogeneous clusters well;
- A greedy algorithm can result in poor local solutions.

The **clustering task is quite difficult** and have a wide variety of applications, so it's almost impossible to build some universal set of rules to select a clustering algorithm - all of them have advantages and disadvantages.

Things become better when you have **some assumptions about your data**, so data analysis can help you with that. What is the approximate number of clusters? Are they located far from each other or do they intersect? Are they similar in shape and density? All that information can help you to solve your task better.

If the number of clusters is unknown, a good initial approximation is the **square root of the number of objects**. You can also first run an algorithm that does not require a number of clusters as a parameter (DBSCAN or Affinity Propagation) and use the resulting value as a starting point.

Learning/generalisation - allows acquiring new knowledge of the environment

Massively parallel processing - allows complex tasks to be performed in a short timeframe

Non-linearity - useful for many real problems

Adaptability - can adapt their topology according to changes in the environment

Robustness and smooth degradation - able to **ignore noise** and irrelevant attributes; handle incomplete information efficiently

Flexibility - large applicability domain

Usability - can be used as "black boxes"; no explicit knowledge of the function to be learned is required

ISLab Synthetic Intelligence Lab

Backpropagation algorithm

It is based on the gradient vector of the error surface which defines the direction of maximum descent - *similar method to the gradient descent*

Important parameter: **learning rate**, which defines the distance one walks

A sequence of these moves leads to a (hopefully global) *minimum*

Training runs for a given number of ***epochs***: defines the number of times each case is trained by the net, and examples are typically divided into ***batches*** (subsets of examples)

Initial net configuration (link weights) is normally **randomly generated**

Stopping criteria: fixed number of epochs, time, or convergence criteria based on a subset of validation examples

Forward propagation - calculated the output value for the input vector and the error committed

Backpropagation - given the error committed this is propagated backwards, adjusting the weights of the connections in the direction of its decrease. It's based on the calculation of the gradient using the chain rule for composed functions

Although there are numerous **variants of neural networks**, each can be defined in terms of:

- An **activation function**, which transforms a node's net input signal into a single output signal to be broadcasted further in the network;
- A **network architecture** (or topology), which describes the **number of nodes** in the model as well as the **number of layers** and manner in which they are connected;
- The **training algorithm** that specifies how connection weights are set in order to inhibit or excite neurons in proportion to the input signal.

The **number of input nodes** is predetermined by the **number of features** in the input data;

The **number of output nodes** is predetermined by the **number of outcomes** to be modeled or the number of class levels in the outcome;

The **number of hidden nodes** is left to the user to decide prior to training the model there is no reliable rule to determine the number of neurons in the hidden layer:
- A **greater number of neurons will result in a model that more closely mirrors the training data**, but this runs a risk of overfitting -> it may generalize poorly to new data
- **Large neural networks can also be computationally expensive** and slow **to train**
- A **short number might not be enough** to model the decision area desired;
- **Values between half and double** of the number of neurons in the input layer should be tested.

Overtraining an ANN *may prevent* generalization by overfitting – The ANN memorizes training cases and not generalization rules - training can be stopped early!

Regularization can be used in a similar way to linear/logistic regression (in ANNs it is called decay)

Probability of overfitting increases if:
- **Few training cases** (sample quality)
- **Too many connections** (net complexity)

Underfitting
This is a model that lacks the necessary complexity to correctly capture the complexity inherent to the problem it is intended to solve. **You can recognize this situation when the error is too large, both in the training cases and in the test (validation) cases**.

Overfitting
This is a model that is using too many parameters and has been trained too much.
Specifically, it has learned to identify exactly each case in the training set, becoming so specific that it cannot generalize to similar images. **You can recognize this situation when the error in the training cases is much smaller than in the test (validation) cases**.

Measures you can take to reduce **overfitting**:
- Adding **more cases** to the training set
- Use architectures that have been shown to generalize well
- Reduce the complexity of the **network architecture**
- Use **data augmentation**
- Add normalization (**Batch Normalization layer**)
- Add Dropout (**Dropout layer**)

ISLab Synthetic Intelligence Lab

Artificial Neural Networks

Strengths
- Classification **accuracy is usually high**, even for complex problems;
- **Distributed processing**, knowledge is distributed through connection weights;
- Robust in handling examples even if they contain errors;
- **Handle redundant attributes well**, since the weights associated with them are usually very small;
- Results can be discrete, real values, or a vector of values (discrete or real).

Weaknesses
- **Difficult to determine the optimal network** topology for a problem;
- **Difficult to use** - have many parameters to define;
- Requires specific data pre-processing;
- Requires **long training time**;
- **Difficult to understand** the learning function (weights);
- **Discovered knowledge** is not readable;
- Do not provide explanations of the results;
- Domain knowledge incorporation is not easy.

- Reinforcement Learning is learning "what to do" in order to maximise a **numerical** reward value:
  - The learner is not told which actions to perform;
  - The learner must discover which actions guarantee the greatest return by trying them out.
- The essential characteristics are:
  - Trial-and-error;
  - Delayed reward.
- Complementary characteristics are:
  - Time;
  - There is a learner, but no teacher!
- Reinforcement learning **is not defined** by the characterization of learning methods (we do not program algorithms to learn);
- Reinforcement learning **is defined** by the characterization of the learning problem (we program the characteristics of the problem to learn about);

- Reinforcement Learning **is not looking** at past experiences to make decisions (CBR, ANN);
- Reinforcement Learning **is looking** at the current state and deciding what to do, predicting the expected future;

We want to obtain a function Q(S,A) (action-value-function) that predicts the best action A in state S in order to maximize a cumulative reward:

Q-learning (off-policy/greedy learner): $$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

New estimation — Learning rate — Discount factor (=0.9)

SARSA (on-policy learner): $$Q(s_t, a_t) \underset{a_{t+1}}{\leftarrow} Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$ s_{t+1}

Q-Learning - The agent is in state 1, performs action 1 and gets reward 1 (r_{t+1});
It sees the maximum reward in state 2 and updates the value of action 1 performed in state 1.

SARSA - The agent is in state 1, performs action 1 and gets reward 1 (r_{t+1});
In state 2, it performs action 2 and obtains reward 2, and then updates the value of action 1 in state 1;

SARSA very much resembles Q-learning, the key difference is that SARSA learns the Q-value based on the action performed by the current policy instead of the greedy policy

Discount factor - Encourages the agent to prefer immediate rewards to delayed ones;

Q-Learning algorithm:

1. **Set** the gamma parameter and environment rewards in the Reward Table.
2. **Initialize** Q-table to zero.
3. **For** each episode:
   Select a random initial state.
   Do While the goal state hasn't been reached.
   - Select one among all possible actions for the current state.
   - Using this possible action, consider going to the next state.
   - Get maximum Q value for this next state based on all possible actions.
   - Compute: *Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]*
   - Update Q-table
   - Set the next state as the current state.

Synthetic Intelligence Lab

Max Voting (classification)

- Each machine learning model makes a vote, the **maximum voted option is the selected**. Simple, yet effective.
- There are two types of max voting:

Hard Voting
Classifier_1 predicts *Class A*
Classifier_2 predicts *Class B*
Classifier_3 predicts *Class B*

2/3 classifier models predict class **B**.
Class B is the **ensemble decision**.

Soft Voting
Classifier_1 predicts *Class A* with the probability of 99%
Classifier_2 predicts *Class A* with the probability of 49%
Classifier_3 predicts *Class A* with the probability of 49%

The average probability of belonging to Class A is (99 + 49 + 49) / 3 = 65.67%.
Thus, ***Class A*** is the **ensemble decision**.

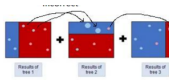
- **Bagging** stands for **Bootstrapped Aggregating**, it has two main steps:
  - Use existing train data and make multiple data instances (**bootstrapping**) – here, you can use the same training samples multiple times.
  - Make multiple models from this bootstrapped data and multiple model outputs. **Aggregate** the results of the model and get the final result.

Synthetic Intelligence Lab

Boosting

First, a **model is run**, and the results are obtained (Model 1), where specific points are classified correctly, and the rest incorrectly. We **give more weight** to the points that are incorrectly classified and rerun the model. Since there are specific points with a higher weight, the model is more likely to include them. We keep **repeating this process**, and multiple models are created, where **each one corrects the errors of the previous one**.

AdaBoost (adaptive boosting):
- Combining decision trees (weak learners)
- Assigning weights to incorrect values
- Sequential tree growing considering past mistakes
- Rebust to overfitting
- Few hyperparameters



Xgboost – gradient boosting - The idea with gradient boosting is to use gradient descent to optimise the parameters of new estimators as estimators are added in boosting.

Taking the best parts of AdaBoost and random forests, adding aditional features:
- Sequential tree growing
- Minimize loss function using gradient descent
- Parallel processing
- Regularization parameter

```
import xgboost as xgb

parameters = {'min_child_weight': [5,6,7], 'colsample_bytree
xgb = xgb.XGBClassifier(objective='reg:squarederror')
clf = GridSearchCV(estimator=xgb, param_grid=parameters, c
best_clf = clf.fit(X_train, y_train)

from sklearn.ensemble import AdaBoostClassifier
```

Synthetic Intelligence Lab

Stacking

The idea of stacking is that we **perform predictions on the train and test dataset with a few models**. This is done on the train and test dataset. We now do not consider the original train and test data. Instead, we consider the new models outputs on the train data as the base train model. The new test data is the models outputs on the test data

Ensemble models work **by combining multiple base learners** into a single strong learner. This helps by decreasing the bias, variance, or improving predictions.

There are two groups of ensemble models :
- **Parallel ensemble models** – The logic employed is to leverage the independence between the base learners. Thus, the mistakes in predictions made by one model are different from those from another independent model. This lets the ensemble model average out the errors. (e.g. Random Forest with independent Decision Trees)
- **Sequential ensemble models** – The logic employed is to leverage the dependence between the base learners. Thus, the mistakes made by the first model are sequentially corrected by the second model and so on. This helps get the most accurate ensemble possible. (e.g. ADABoost is sequencial)
  For example, the AdaBoost Ensemble Model is sequential