Machine Learning methods for the electricity market

Federico Minutoli - Gianvito Losapio

90498 - Machine Learning

Final project

MSc in Computer Science

a.y. 2019 - 2020

January 20, 2019

Outline

Mix project of types 1 and 2: same problem proposed by the IREN company during C1A0 Hackathon.

- Problem description
- Data processing } Type 2
- Random Forests (by Gianvito Losapio)
- Gradient Boosting (by Federico Minutoli) } Type 1
- Experiments and results
- Conclusions and future works

Problem description

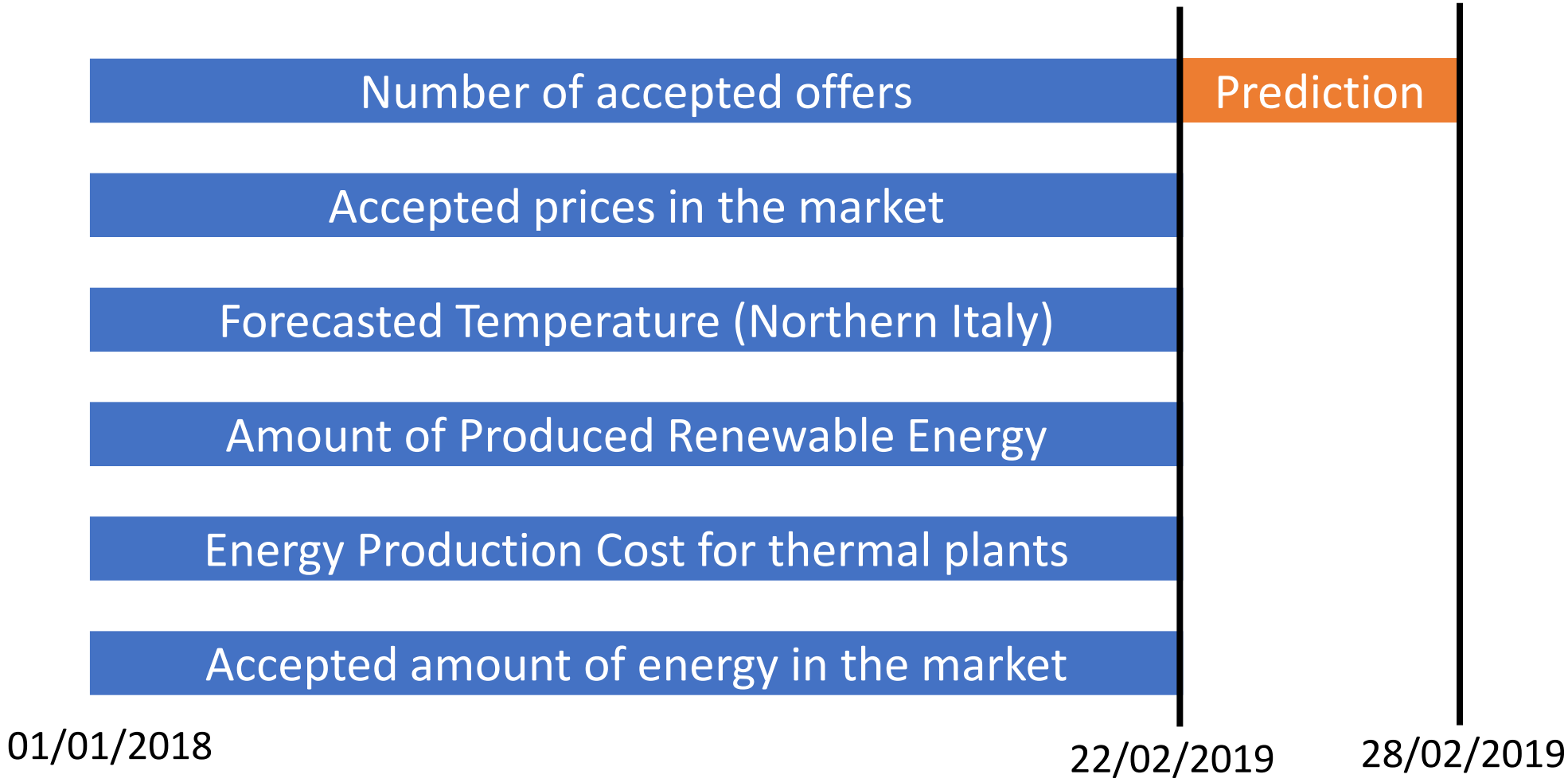
- The Day-Ahead Market (**MGP =** Mercato Giorno Prima), is the market for the trading of electricity supply offers and demand bids for each hour of the next day in Italy. All electricity operators may participate in the MGP.

- Participants submit offers (bids) where they specify the quantity and the minimum (maximum) price they are willing to sell (purchase). The **accepted offers** are those with a submitted price not larger than the marginal clearing price (obtained by Gestore Mercati Energetici from the intersection of the demand and supply curves). The accepted bids are those with a submitted price not lower than the PUN (Prezzo Unico Nazionale – national single price, agreed daily)

Our **goal** is to forecast 28 different time series:
the number of daily accepted offers for the week 22-28th February 2019 for a specific list of participants, identified by IREN itself

Data processing

- We are given some **hourly data** referred to the previous 14 months widespread across several tables



Data processing

- Data cleaning (keep only useful data) and format conversion (homogeneous dates, measure units)

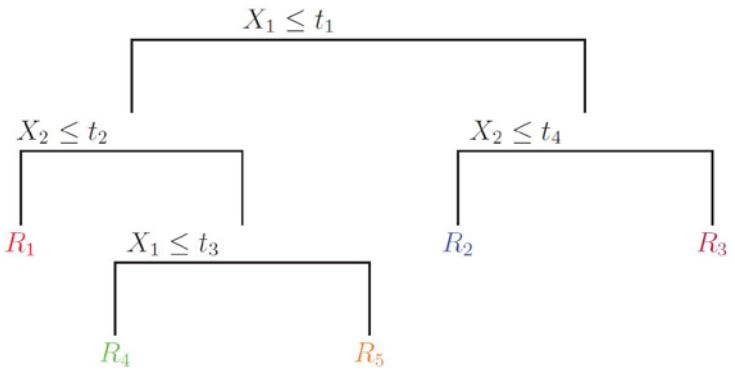
- Daily statistics of each feature (mean)

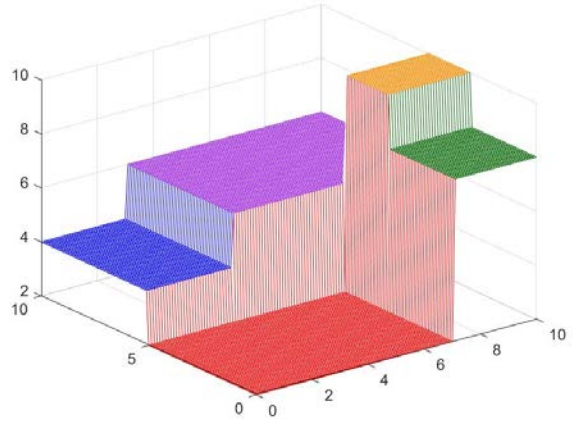
- Time-series analysis: almost all the time series present spikes in the ACF (*Autocorrelation function*) plot over a *7-day* span. It may be a strong indicator of a weekly seasonality → lag feature with rolling windows of 3,7 days (mean)

- Output shifting of 7 days to give the idea of future

CART models

- Basic building block of both algorithms





- Growing procedure?
- Local model?
- Tree size?

Random Forest (by Gianvito Losapio)

ERM $\Theta^* = \arg\min_{\Theta} \sum_{i=1}^{n} \ell\big(y_i, T(x;\Theta)\big)$

$\Theta = \{\mathcal{D}_m, \gamma_m\}_{m=1}^{M}$

$\mathcal{D} = \{x_i, y_i\}_{i=1}^{n}$

$\mathcal{D}_L(j,t) = \{(x_i, y_i) \,|\, x_{ij} \leq t\}$ and $\mathcal{D}_R(j,t) = \{(x_i, y_i) \,|\, x_{ij} > t\}$

Greedy solution:

$$\min_{j,t}\left[\underbrace{\min_{\gamma_L}\frac{1}{|\mathcal{D}_L(j,t)|}\sum_{y_i\in\mathcal{D}_L(j,t)}(y_i-\gamma_L)^2}_{\text{cost}(\mathcal{D}_L)} + \underbrace{\min_{\gamma_R}\frac{1}{|\mathcal{D}_R(j,t)|}\sum_{y_i\in\mathcal{D}_R(j,t)}(y_i-\gamma_R)^2}_{\text{cost}(\mathcal{D}_R)}\right]$$

Best split:

$$(j^*, t^*) = \arg\min_{j=1,\dots,d}\min_{t\in\mathcal{T}_j}\frac{|\mathcal{D}_L|}{|\mathcal{D}|}\text{cost}(\mathcal{D}_L) + \frac{|\mathcal{D}_R|}{|\mathcal{D}|}\text{cost}(\mathcal{D}_R)$$

- Bootstrap aggregating
- Feature bagging

Random Forest (by Gianvito Losapio)

```
class RandomForestRegressor():
    """
    params: dict
        Set of building options

        max_depth: int, default=None
        The maximum depth of the tree. If None, then nodes are expanded
        until all leaves are pure or until all leaves contain less than
        min_samples_split samples.

        n_estimators: int, default=10
        The number of trees in the forest.

        max_features: int, default=None
        The number of features to consider when looking for
        the best split

        min_samples_split: int, default=2
        The minimum number of samples required to split an internal node.

        min_samples_leaf: int, default=1
        The minimum number of samples required to be at a leaf node.
        A split point at any depth will only be considered if it produces
        at least min_samples_leaf training samples in each of the left
        and right branches.

        min_impurity_decrease: float, default=0.0
        A node will be split if this split induces a decrease of the
        objective function greater than or equal to this value.


    """
```

CV

```
class Tree():
    """
    Decision tree class for regression.
```

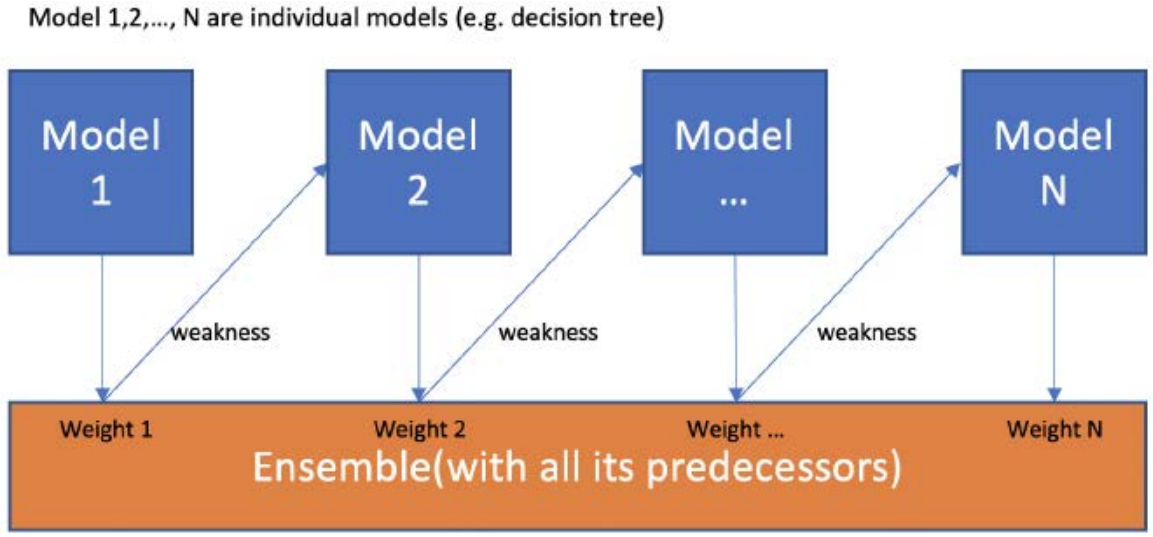
```
class Node:
    """Node class for the regression tree model

def build(self, X_instances, Y_instances, depth, params, file=None):
    """Greedy algorithm to build the node and create its
```

Best split strategy implemented here

Gradient boosting (by Federico Minutoli)

$$f(x) = \sum_{m=0}^{M} \beta_m h_m(\mathbf{x})$$ Adaptive basis-function model



1. Start with an initial guess $F_0(x)$

2. for m = 1 to M:

$$(\beta_m, a_m) = \arg\min_{\beta,a} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta h(x_i; a)$$

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=f_{m-1}(x)}$$

Fit a regression tree

$$h(x; \{R_{l,m}\}_{l=1}^{L}) = \sum_{l=1}^{L} \bar{y}_{l,m} I(x \in R_m)$$

Compute pseudo-residuals

GBM pipeline

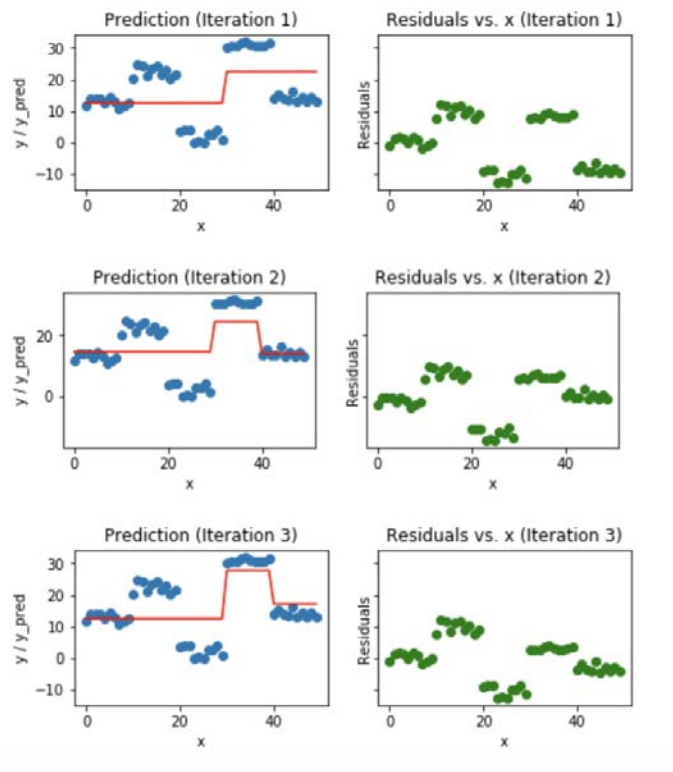
Compute weights

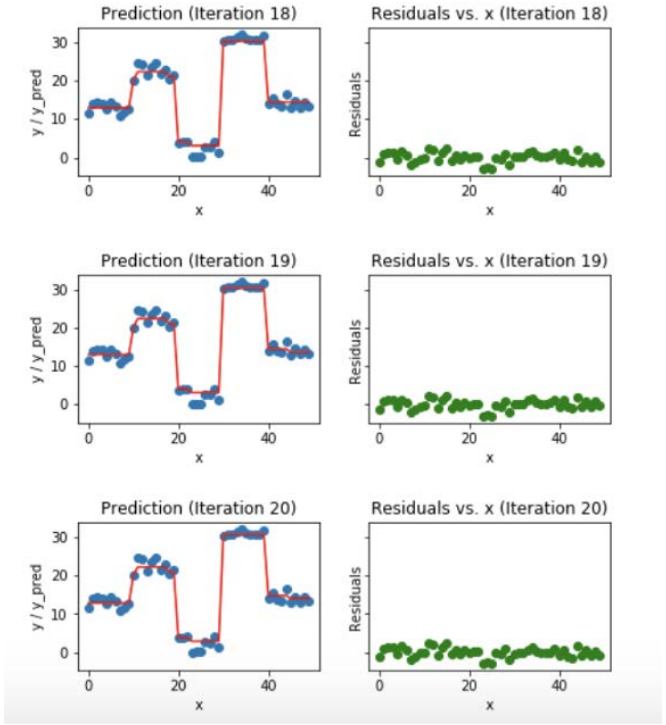
$$f_m(x) = f_{m-1}(x) + \nu \sum_{j=1}^{T} \gamma_{jm} \mathbb{1}_{\{x_i \in R_{jm}\}}$$

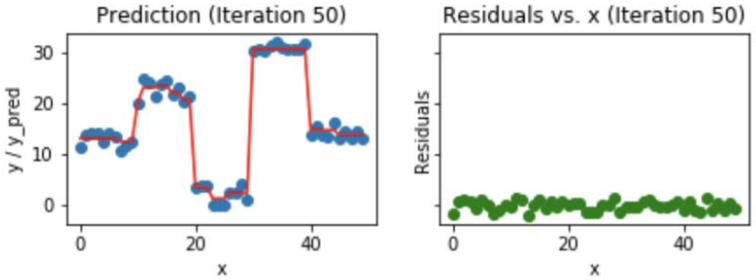
Update

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma h_m(x_i))$$

Gradient boosting (by Federico Minutoli)







- Learning rate

Gradient boosting (by Federico Minutoli)

OF $\phi = \sum_i L(y_i, f_m(x_i)) + \sum_k \Omega(f_k)$, $\Omega(f) = \gamma T + \frac{1}{2}\lambda ||w||^2$

$$\mathcal{D} = \{x_i, y_i\}_{i=1}^n$$

...

$$w_j^* = \arg\min_{w_j} \sum_{i \in I_j} g_i w_j + \frac{1}{2}\left(\sum_{i \in I_j} h_i + \lambda\right) w_j^2 = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

$$\tilde{\phi_m} = \sum_{j=1}^{T} \left[\left(\sum_{i \in I_j} g_i\right) w_j + \frac{1}{2}\left(\sum_{i \in I_j} h_i + \lambda\right) w_j^2\right] + \gamma T$$

Greedy optimal value:

$$\tilde{\phi_m}(q) = -\frac{1}{2}\sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \lambda T$$

$$I_j = \{i \,|\, q(x_i) = j\}$$
$$I = I_L \cup I_R$$

Information gain:

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda}\right] - \gamma$$

- Feature subsampling

Gradient boosting (by Federico Minutoli)

```python
class Node:
    """Node class for the GBDT

    Attributes
    ----------
    leaf: bool
        True if the node is a leaf

    child_left: Node
        Left child node

    child_right: Node
        Right child node

    split_feature_id: int
        ID of the feature that led to the
        best split from this node

    split_val: int
        Instance of the node associated to the
        feature whose ID is split_feature_id

    weight: int
        Weight associated to each leaf node when
        its depth exceeds the tree's max depth

    """


class Tree():
    """Regression tree for Ensemble learning"""
```

```python
class GBDT():
    """Gradient boosting class for regression built upon
    CART trees that descend following an L2-gradient.
```

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i,\ H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ ***to*** m **do**
  $G_L \leftarrow 0,\ H_L \leftarrow 0$
  for j *in sorted*$(I,$ *by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j,\ H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L,\ H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

Experiments and results

- Divide and Conquer strategy , 5-Fold Cross validation

Table 1: Root Mean Squared Error (RMSE)

Time series		Test set performance		
participant	technology	Random Forest	GBDT	LightGBM
participant_116	ccgt	0.0	0.0	1.0
participant_116	ccgt400	0.654	1.647	0.0
participant_116	ccgt800	0.755	0.654	0.755
participant_116	pump	0.0	1.0	0.0
participant_116	reservoir	0.0	2.0	0.0
participant_120	ccgt800	0.0	0.0	0.0
participant_146	pump	0.534	0.534	0.377
participant_146	reservoir	0.925	0.925	1.0
participant_158	ccgt400	0.0	1.0	0.0
participant_158	ccgt800	0.755	0.925	0.845
participant_158	pump	0.534	0.845	0.845
participant_158	reservoir	0.0	1.0	0.0
participant_176	ccgt800	0.0	0.845	0.377
participant_21	reservoir	0.0	0.0	0.0
participant_60	ccgt400	0.0	1.0	0.0
participant_75	ccgt400	0.925	0.845	0.534
participant_75	ccgt800	0.845	0.845	0.0
participant_87	ccgt	0.377	0.925	0.377
participant_87	ccgt400	0.0	2.329	0.534
participant_89	ccgt400	1.133	0.654	1.133
participant_89	ccgt800	0.377	0.845	0.534
participant_89	coal	0.0	1.0	0.0
participant_89	reservoir	0.0	1.0	0.0
participant_96	ccgt400	0.377	0.925	0.534
participant_96	ccgt800	0.925	0.0	0.654
participant_96	coal	0.654	2.171	0.534
participant_96	pump	0.925	1.463	0.925
participant_96	reservoir	1.309	1.253	0.654

Global RMSE	Random Forest	GBDT	LightGBM
Validation	0.774	1.338	0.583
Test	0.601	1.113	0.562

Conclusions

- Time-series analysis is key in the early stages
- Lag features design

- Random forests and Gradient boosting
  - Pros:
    - Powerful methods to tackle both regression and classification problems
    - Easy to implement and available as off-the-shelf modules
  - Cons:
    - High number of parameters has to be cross-validated. This is a really costly procedure and hinders the interpretability of the model
    - Broadly used with a black-box approach