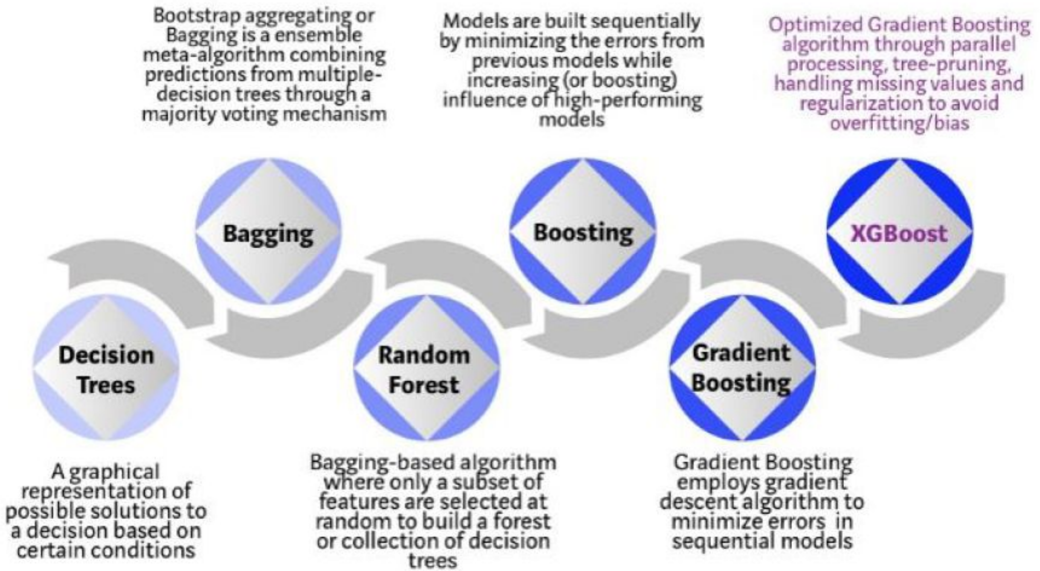
XGBoost
(Extreme Gradient Boost)

Reminder

The evolution of tree-based methods



Evolution of XGBoost Algorithm from Decision Trees

Reminder

Example: L1 and L2 regularization of a polynomial regression

"let the model itself evaluate the importance of features as part of its training process"

Lasso

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N}\Big(y_i - \sum_{j=0}^{M} x_{ij}W_j\Big)^2 + \lambda\sum_{j=0}^{M}|W_j|$$

Ridge

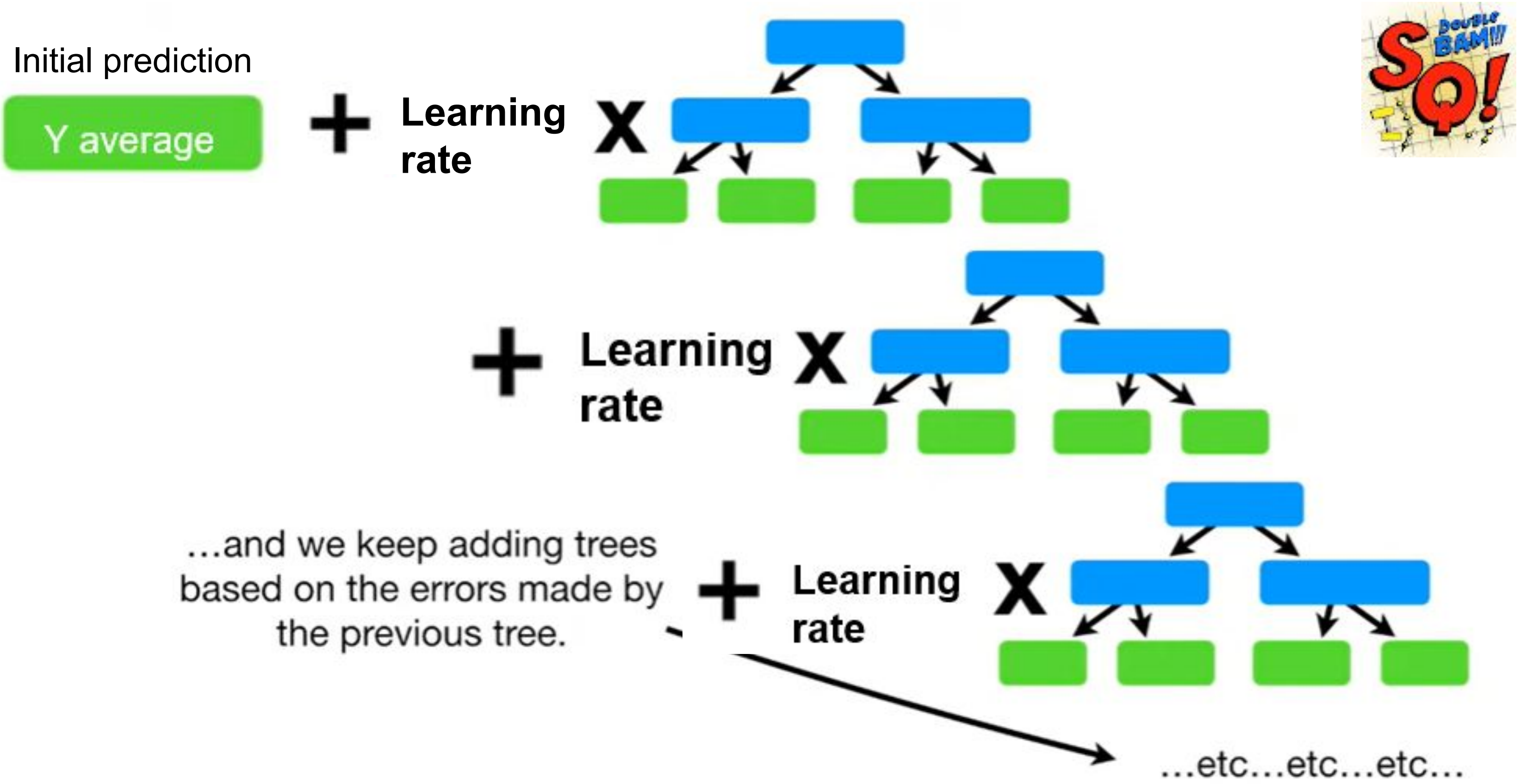
L2 Regularization

$$\text{Cost} = \underbrace{\sum_{i=0}^{N}\Big(y_i - \sum_{j=0}^{M} x_{ij}W_j\Big)^2}_{\text{Loss function}} + \underbrace{\lambda\sum_{j=0}^{M}W_j^2}_{\text{Regularization Term}}$$

Elastic-net

$$L = \underbrace{\boxed{\sum(y-\hat{y})^2}}_{\text{Sum of Squared Residuals}} + \underbrace{\boxed{\alpha\sum w^2}}_{\text{Penalty (RIDGE)}} + \underbrace{\boxed{\alpha\sum|w|}}_{\text{Penalty (LASSO)}}$$

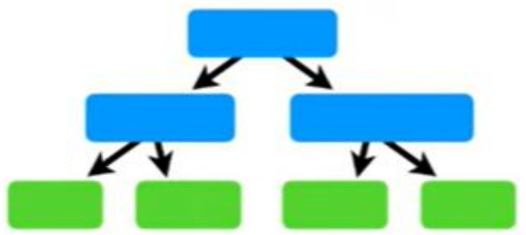
Reminder? Gradient Boosting



Extreme Gradient Boost

Initial prediction

Anything
(default 0.5)



1) How to split the data?

For different treshold, build a tree:

- Calculate Similarity scores based on residuals
- Calculate the gain that reflects how much better the leaves cluster similar residual than the root

 Keep the treshold with the highest gain

XGBoost: Similarity Scores

In regression

$$\text{Similarity Score} = \frac{\text{Sum of Residuals, Squared}}{\text{Number of Residuals} + \lambda}$$

In classification

$$\text{Similarity Score} = \frac{(\sum \text{Residual}_i)^2}{\underbrace{\sum [\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)]}_{\text{Cover}} + \lambda}$$

XGBoost: Gain

- How much better the leaves cluster similar residual than the root

$$\text{Gain} = \text{Left}_{\text{Similarity}} + \text{Right}_{\text{Similarity}} - \text{Root}_{\text{Similarity}}$$

 Keep treshold with the highest gain

- How to prune the tree? (reduce model complexity to prevent overfitting)
  - User set the parameter Gamma

$$\text{Gain} - \gamma = \begin{cases} \text{If positive, then do not prune.} \\ \text{If negative, then prune.} \end{cases}$$

XGBoost: Hyperparameters

- **λ (lambda):** L2 regularization on leaf weights.
- **γ (gamma):** Model complexity. Higher γ means stronger pruning — fewer, simpler trees.
- **min_child_weight:** Prevents leaves with very few observations, particularly relevant for classification.

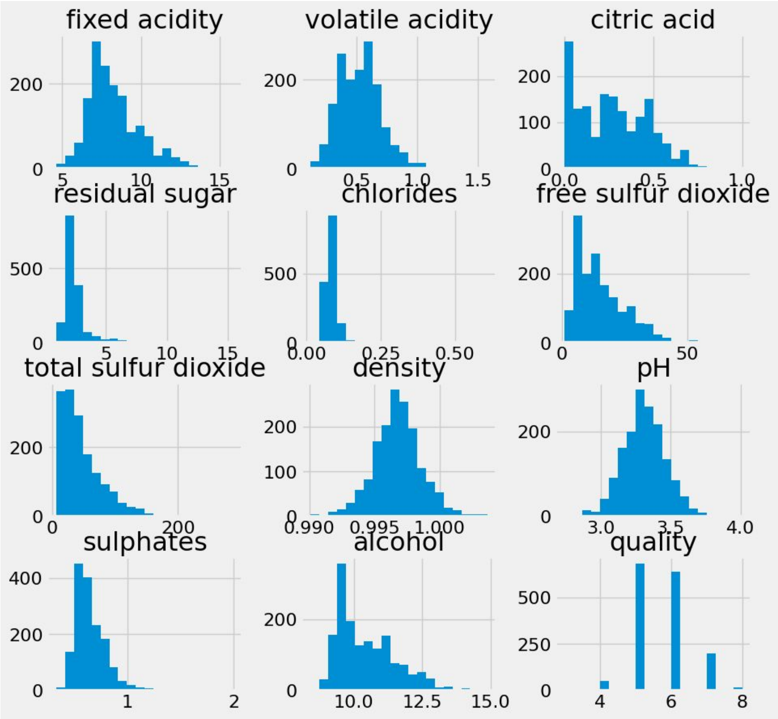
Why using XGBoost?

- Regularized boosting (bias–variance control)
- Second-order optimization (fast, accurate)
- Captures nonlinearities and interactions
- Handles missing and sparse data
- Works for regression and classification

EXAMPLES

Regression & Classification

Using XG-Boost on a red/white wine Dataset



Xg_Boost for Regression

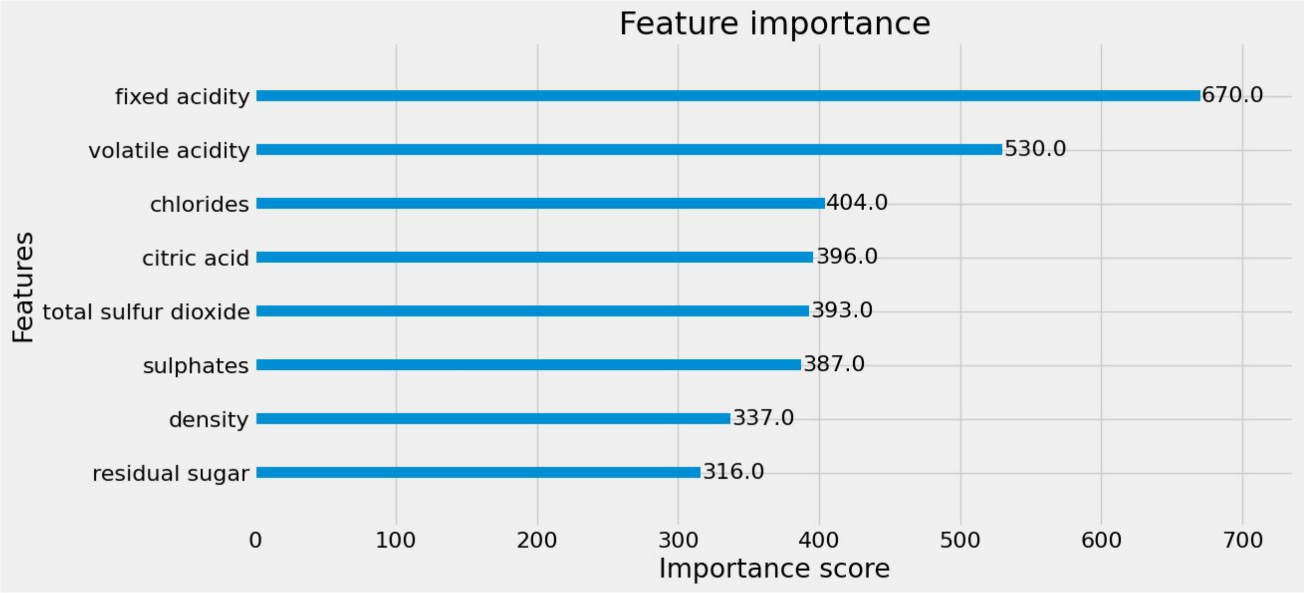
80-20 Train-Test split

Deploying grid search to find the optimal Hyperparameters

	param_learning_rate	param_max_depth	param_n_estimators	mean_test_score	rank_test_score	RMSLE
16	0.015	6	600	-0.078066	1	0.279403
15	0.015	6	500	-0.078078	2	0.279424
6	0.010	6	500	-0.078083	3	0.279433
7	0.010	6	600	-0.078091	4	0.279448
8	0.010	6	700	-0.078242	5	0.279718
17	0.015	6	700	-0.078287	6	0.279799
14	0.015	5	700	-0.078299	7	0.279820
13	0.015	5	600	-0.078485	8	0.280152
12	0.015	5	500	-0.078622	9	0.280395
5	0.010	5	700	-0.078940	10	0.280962
4	0.010	5	600	-0.079035	11	0.281132
3	0.010	5	500	-0.079188	12	0.281403
2	0.010	4	700	-0.080395	13	0.283540
11	0.015	4	700	-0.080585	14	0.283875
9	0.015	4	500	-0.080630	15	0.283955
10	0.015	4	600	-0.080671	16	0.284026
1	0.010	4	600	-0.080680	17	0.284043
0	0.010	4	500	-0.081195	18	0.284947

Regression Result

Feature Importance



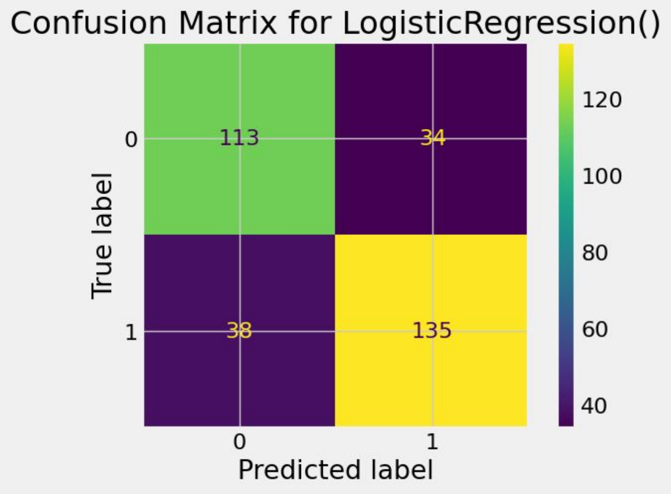
Using XG-Boost for classification

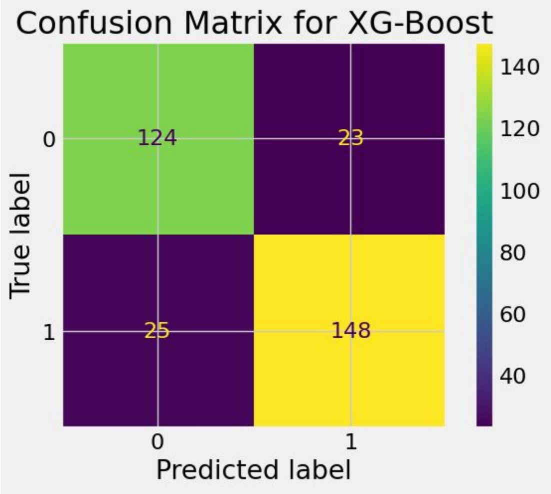
- Try to classify wine quality Quality > 5 = Good
- Splitting the dataset in 80/20 train-test ratio

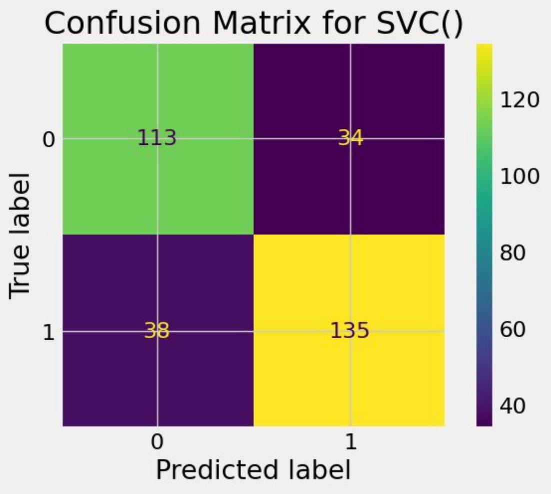
Train three different Models SVC, XG-Boost, Linear Regression

```
LogisticRegression() :
Training Accuracy :  0.7386369776546466
Validation Accuracy :  0.7745271519012229
Training time: 0.0285947322845459 seconds

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              feature_weights=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=None,
              n_jobs=None, num_parallel_tree=None, ...) :
Training Accuracy :  1.0
Validation Accuracy :  0.8495143722228776
Training time: 0.15081286430358887 seconds

SVC() :
Training Accuracy :  0.7762284049769865
Validation Accuracy :  0.7745271519012229
Training time: 0.06345152854919434 seconds
```

Train three different Models SVC, XG-Boost, Linear Regression







Summary

Regularized boosting: balances bias-variance trade-off

Second-order optimization: fast and accurate learning

Captures nonlinearities and feature interactions

Robust to missing and sparse data; supports regression & classification

References

XGBoost Part 1 (of 4): Regression
XGBoost Partie 2: Arbres XGBoost pour la classification
XGBoost Part 1 (of 4): Regression
A Guide on XGBoost hyperparameters tuning