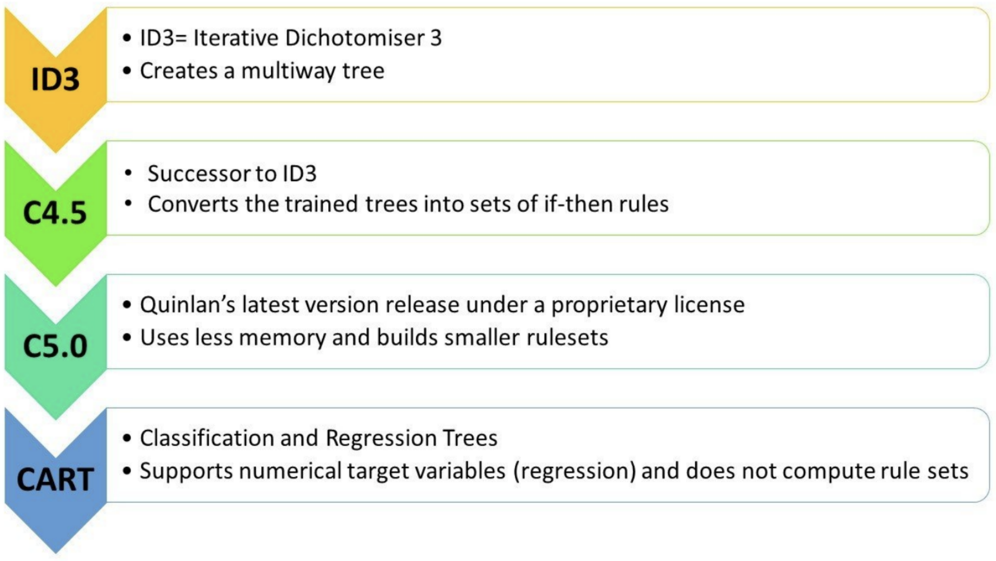
Week 8

Decision Trees

- non-parametric supervised learning method
- can learn classification and regression models
- black-box estimator

Tree Algorithms


ID3
• ID3= Iterative Dichotomiser 3
• Creates a multiway tree
C4.5
• Successor to ID3
• Converts the trained trees into sets of if-then rules
C5.0
• Quinlan's latest version release under a proprietary license
• Uses less memory and builds smaller rulesets
CART
• Classification and Regression Trees
• Supports numerical target variables (regression) and does not compute rule sets

Sklearn Implementation of Trees

scikit-learn uses an **optimized version of the CART algorithm**; however, it **does not support categorical variable**s for now

- Classification - sklearn.tree.DecisionTreeClassifier
- Regression - sklearn.tree.DecisionTreeRegressor

Both these estimators have the same set of parameters (**except for criterion** used for tree splitting).

Parameters

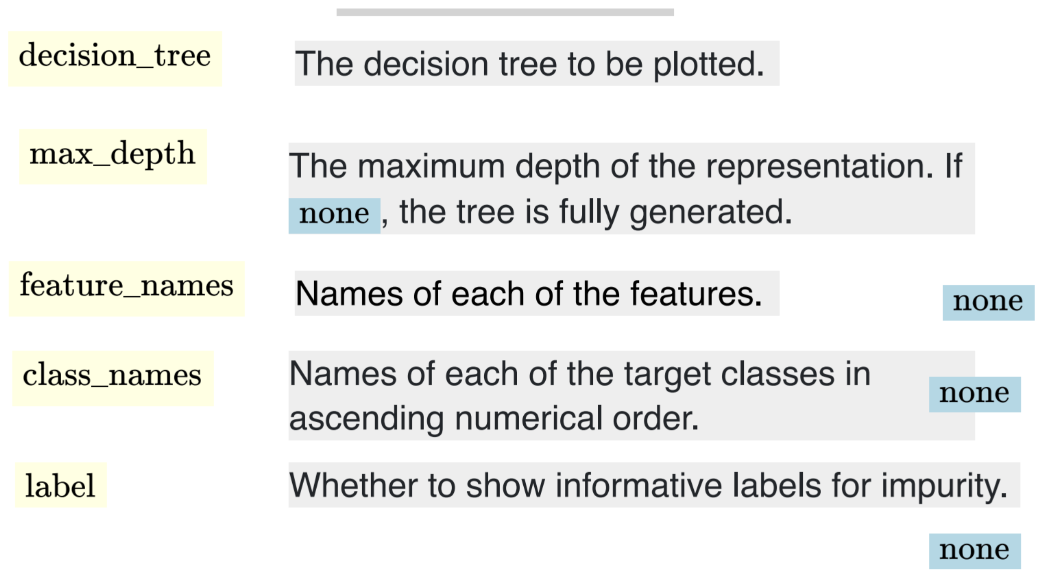
- **splitter**
  - Strategy for splitting at each node.
  - *best*, random
- **max_depth** (int)
  - Maximum depth of the tree
  - When *None*, the tree expanded until all leaves are pure or they contain less than samples.
- **min_samples_split** (int, float)
  - The minimum number of samples required to split an internal node

    - Default: *2*
- **min_samples_leaf** (int, float)
    - The minimum number of samples required to be at a leaf node
    - Default: *1*
- **criterion**
    - Classification
        - *gini*
        - entropy
    - Regression
        - *squared_error*
        - friedman_mse
        - absolute_error
        - poisson

Note: Default values are in italics

Visualisation

💡 sklearn.tree.plot_tree



decision_tree	The decision tree to be plotted.	
max_depth	The maximum depth of the representation. If none, the tree is fully generated.	
feature_names	Names of each of the features.	none
class_names	Names of each of the target classes in ascending numerical order.	none
label	Whether to show informative labels for impurity.	none

Avoiding overfitting of trees

- **Pre-pruning**

  Uses hyper-parameter search like GridSearchCV for finding the best set of parameters.

- **Post-pruning**

  First grows trees without any constraints and then uses cost_complexity_pruning with max_depth and min_samples_split

Tips for practical usage

- Decision trees tend to overfit data with a large number of features. Make sure that we have the right ratio of samples to number of features.
- Perform dimensionality reduction (PCA, or Feature Selection) on a data before using it for training the trees. It gives a better chance of finding discriminative features.
- Visualize the trained tree by using max_depth=3 as an initial tree depth to get a feel for the fitment and then increase the depth.
- Balance the dataset before training to prevent the tree from being biased toward the classes that are dominant.
- Use min_samples_split or min_samples_leaf to ensure that multiple samples influence every decision in the tree, by controlling which splits will be considered.
  - A very small number will usually mean the tree will overfit.
  - A large number will prevent the tree from learning the data.

Voting estimators

💡 sklearn.ensemble.VotingClassifier, sklearn.ensemble.VotingRegressor

Working

A Voting Classifier is a machine learning model that trains on an ensemble of numerous models and predicts an output (class) based on their highest probability of chosen class as the output.

It simply aggregates the findings of each classifier passed into Voting Classifier and predicts the output class based on the highest majority of voting. The idea is instead of creating separate dedicated models and finding the accuracy for each them, we create a single model which trains by these models and predicts output based on their combined majority of voting for each output class.

Parameters and Functions

Both these estimators take the following common parameters:

- base_estimator
- weights

Both these estimators implement the following functions:

- fit
- fit_transform
- predict
- score

VotingClassifier takes an additional argument 'voting':

- **hard:** In hard voting, the predicted output class is a class with the highest majority of votes i.e the class which had the highest probability of being predicted by each of the classifiers
- **soft:** In soft voting, the output class is the prediction based on the average of probability given to that class.

Bagging estimators

💡 sklearn.ensemble.BaggingClassifier, sklearn.ensemble.BaggingRegressor

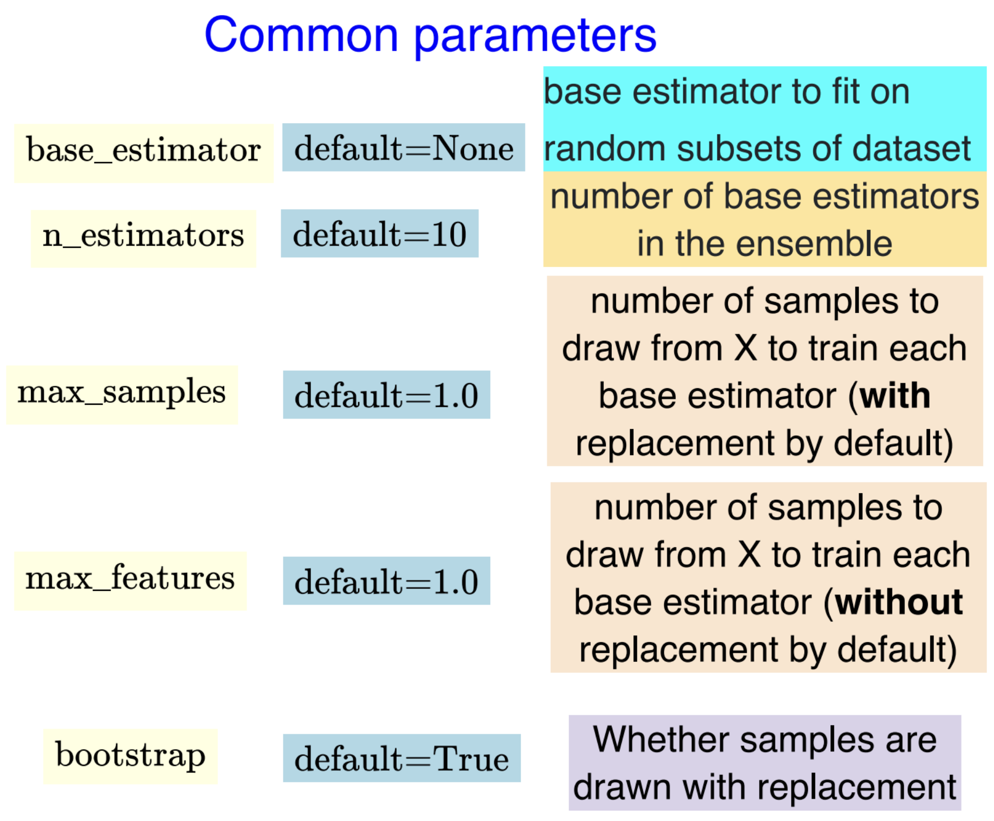
A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it.Each base classifier is trained in parallel with a training set which is generated by randomly drawing, with replacement, N examples(or data) from the original training dataset – where N is the size of the original training set. Training set for each of the base classifiers is independent of each other. Many of the original data may be repeated in the resulting training set while others may be left out.

Bagging reduces overfitting (variance) by averaging or voting, however, this leads to an increase in bias, which is compensated by the reduction in variance though.

How Bagging works on training dataset?

Since Bagging resamples the original training dataset with replacement, some instance(or data) may be present multiple times while others are left out.

Parameters



Common parameters

base_estimator	default=None	base estimator to fit on random subsets of dataset
n_estimators	default=10	number of base estimators in the ensemble
max_samples	default=1.0	number of samples to draw from X to train each base estimator (**with** replacement by default)
max_features	default=1.0	number of samples to draw from X to train each base estimator (**without** replacement by default)
bootstrap	default=True	Whether samples are drawn with replacement

bootstrap_features	default=False	Whether features are drawn with replacement
oob_score	default=False	Whether to use out-of-bag samples to estimate generalization error

Random Forest

sklearn.ensemble.RandomForestClassifier,
sklearn.ensemble.RandomForestRegressor

Parameters

Bagging Parameters

- The **number of trees** are specified by ***n_estimators***:
    - Default #trees for classification = 10
    - Default #trees for regression = 100
- ***bootstrap*** specifies w**hether to use bootstrap samples** for training.
    - True: bootstrapped samples are used.
    - False : whole dataset is used.
- **oob_score** specifies whether to use out-of-bag samples for estimating generalization error.

  It is only available when bootstrap = true.
- **max_samples** specifies the **number of samples** to be drawn while bootstrapping.
    - None: Use all samples in the training data.
    - int: Use samples from the training data.
    - float : Use max_samples * total no of samples from training data
      The value should be between 0 and 1.

- **random_state** controls randomness of features and samples selected during bootstrap.
- The number of features to be considered while splitting is specified by **max_features**.

Value	**max_features**
int	value specified
float	value * # features
auto	sqrt(#features)
sqrt	sqrt(#features)
log2	log2(#features)
None	#features

Decision Tree Parameters

- The **criteria for splitting the node** is specified through ***criterion***.
  - Default for classification: *gini*
  - Default for regression: *squared_error*
- The depth of the tree is controlled by ***max_depth***. Default: None
- We will continue to split the internal node until they contain min_samples_split samples.
  - Whenever it is specified as an integer, then it is considered as a number.
  - Whenever it is specified as a float, and the *min_samples_splits* is calculated as $\text{min\_samples\_splits} \times n$.
- The tree growth can also be controlled by ***min_impurity_decrease*** parameter.
  - A node will be split if it reduces impurity at least by the value specified in this parameter.
- The complexity of tree can also be controlled by ***ccp_alpha*** parameter through minimal cost complexity pruning procedure.

Trained random forest estimators

- ***estimators_*** member variable contains a collection of fitted estimators.
- ***feature_importances_*** member variable contains a list of important features.

Training and Inference

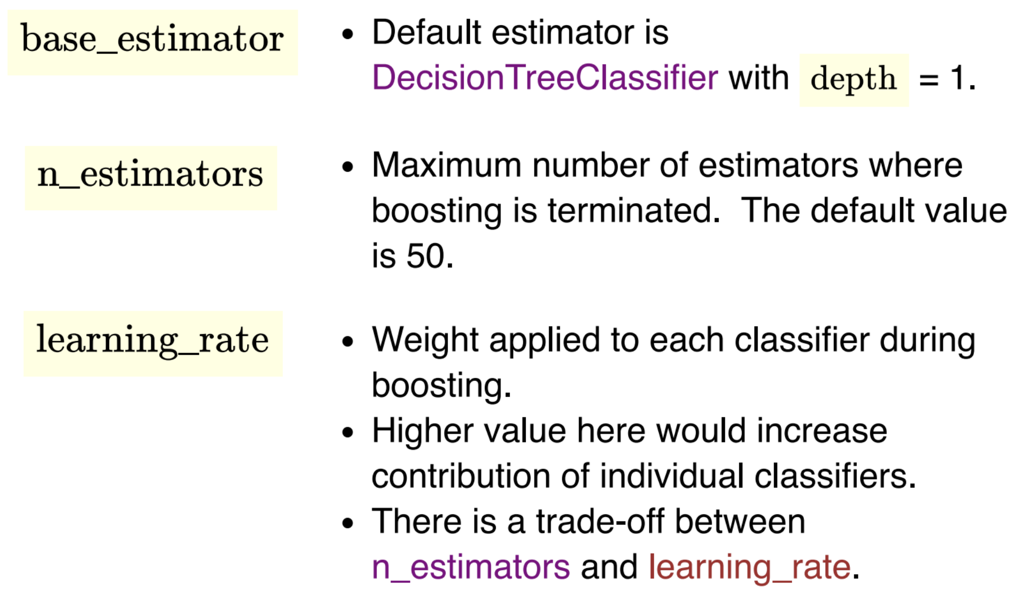
- ***fit*** builds forest of trees from the training dataset with the specified parameters.
- ***decision_path*** returns decision path in the forest.
- ***predict*** returns class label in classification and output value in regression.
- ***predict_proba*** and ***predict_log_proba*** returns probabilities and their logs for classification set up.

Boosting

Boosting is an ensemble modeling technique that attempts to build a strong classifier from the number of weak classifiers. It is done by building a model by using weak models in series. Firstly, a model is built from the training data. Then the second model is built which tries to correct the errors present in the first model. This procedure is continued and models are added until either the complete training data set is predicted correctly or the maximum number of models are added.

AdaBoost

💡 sklearn.ensemble.AdaBoostClassifier



base_estimator	• Default estimator is DecisionTreeClassifier with depth = 1.
n_estimators	• Maximum number of estimators where boosting is terminated. The default value is 50.
learning_rate	• Weight applied to each classifier during boosting. • Higher value here would increase contribution of individual classifiers. • There is a trade-off between n_estimators and learning_rate.

💡 sklearn.ensemble.AdaBoostRegressor

base_estimator	• Default estimator is DecisionTreeRegressor with depth = 3.
n_estimators	• Maximum number of estimators where boosting is terminated. The default value is 50.
learning_rate	• Weight applied to each regressor at each boosting iteration. • Higher value here would increase contribution of individual regressor. • There is a trade-off between n_estimators and learning_rate.

The main parameters to tune to obtain good results are

- **n_estimators** and
- Complexity of the base estimators (e.g. its depth **max_depth** or **min_samples_split**).

Attributes

base_estimator_	Base estimator of ensemble.
estimators_	Collection of fitted sub-estimators.
estimator_weights_	Weights for each estimator in ensemble.
estimator_errors_	Errors for each estimator in ensemble.

Gradient Boosting Estimators

💡 sklearn.ensemble.GradientBoostingClassifier, sklearn.ensemble.GradientBoostingRegressor

There are two most important parameters of these estimators:

- n_estimators
- learning_rates

sklearn.ensemble.GradientBoostingClassifier supports both binary and multiclass classification.