[illegible]

CSC 462: Machine Learning

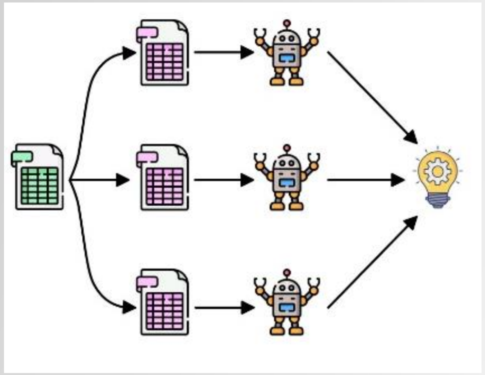
7.5 Ensemble Learning

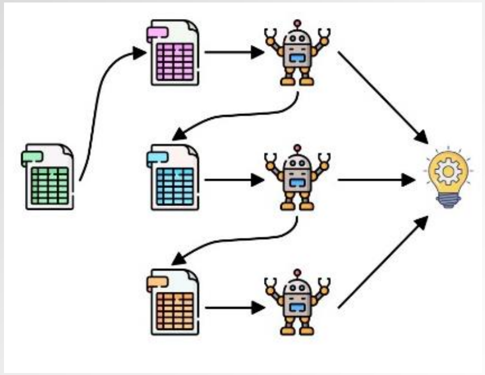
Dr. Sultan Alfarhood

7.5 Ensemble Learning

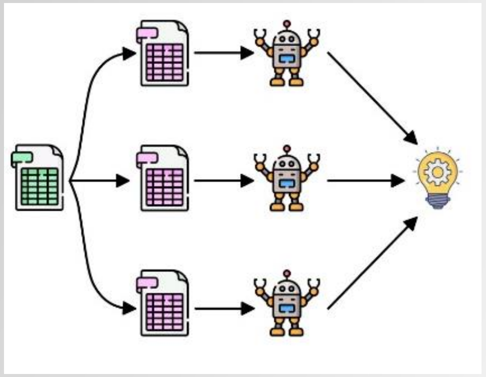
- Ensemble learning **combines** several models to improve overall accuracy of ML algorithms

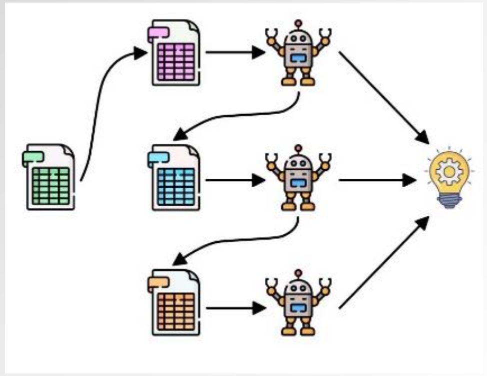
- Training a large number of low-accuracy models and then combining the predictions given by those weak models to obtain a high-accuracy meta-model





7.5 Ensemble Learning

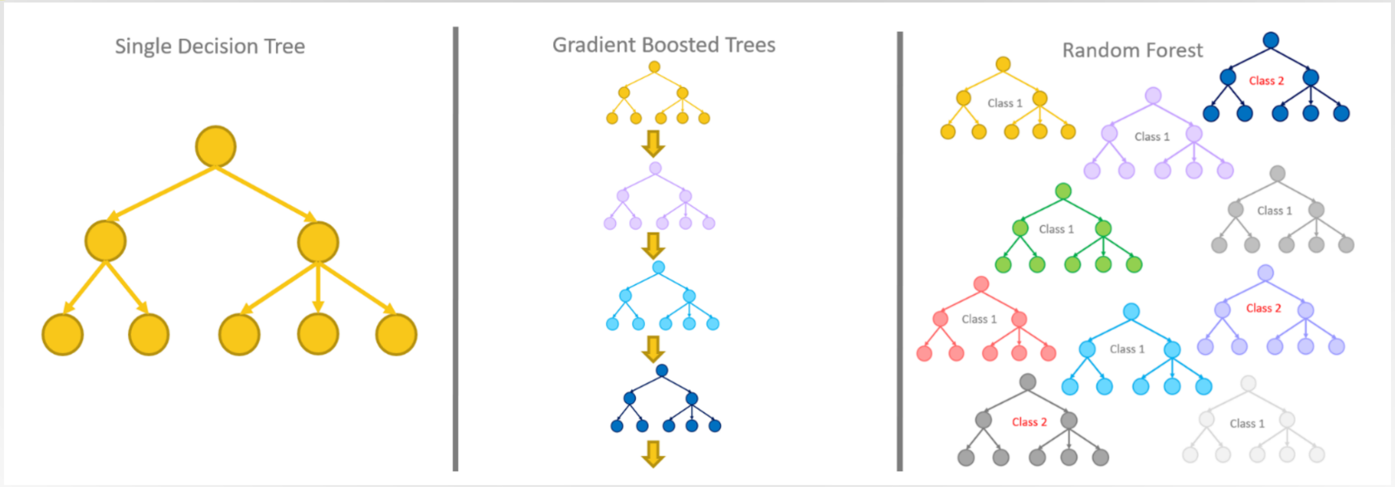




- **Weak learners** are learning algorithms that cannot learn complex models
  - Typically **fast** at the training and at the prediction time
  - The most frequently used weak learner is a **decision tree** learning algorithm (stop splitting the training set after just a few iterations)
    - If the trees are not identical and each tree is at least slightly better than random guessing, then we can obtain high accuracy by combining a large number of such trees

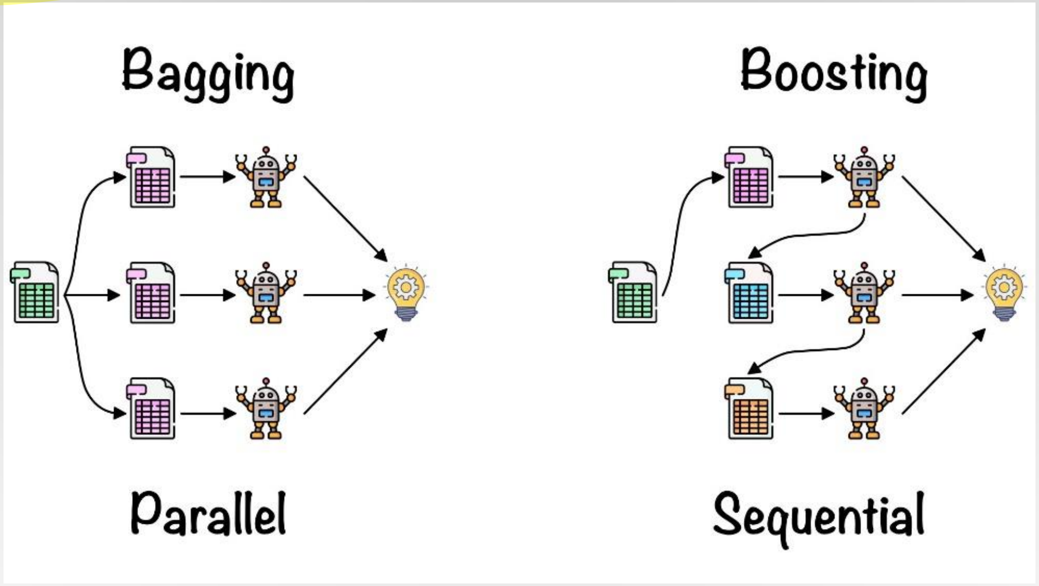
7.5 Ensemble Learning

- Most widely used and effective ensemble learning algorithms are
  - Random Forest
  - Gradient Boosting
  - XGBoost
  - LightGBM
  - CatBoost



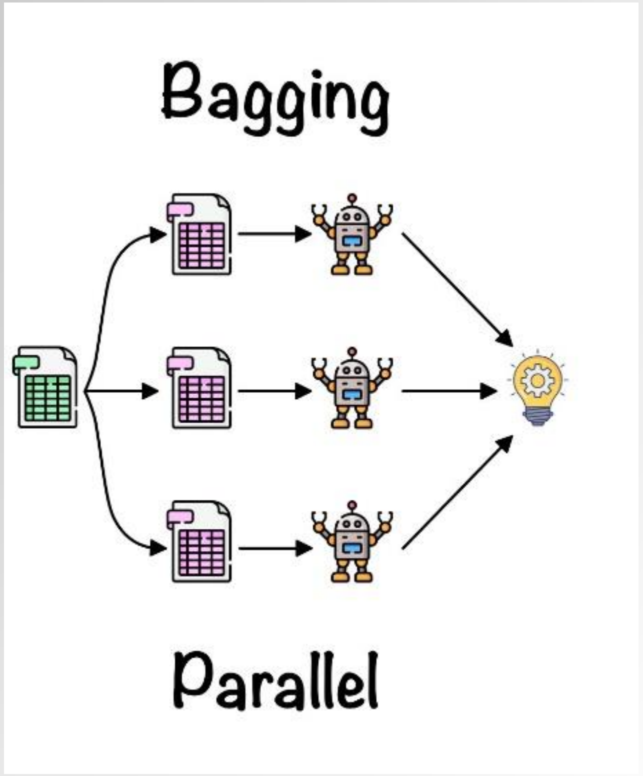
7.5 Ensemble Learning

- Ensemble learning paradigms
  - Bagging (Used in Random Forest)
  - Boosting (Used in Gradient Boosting)

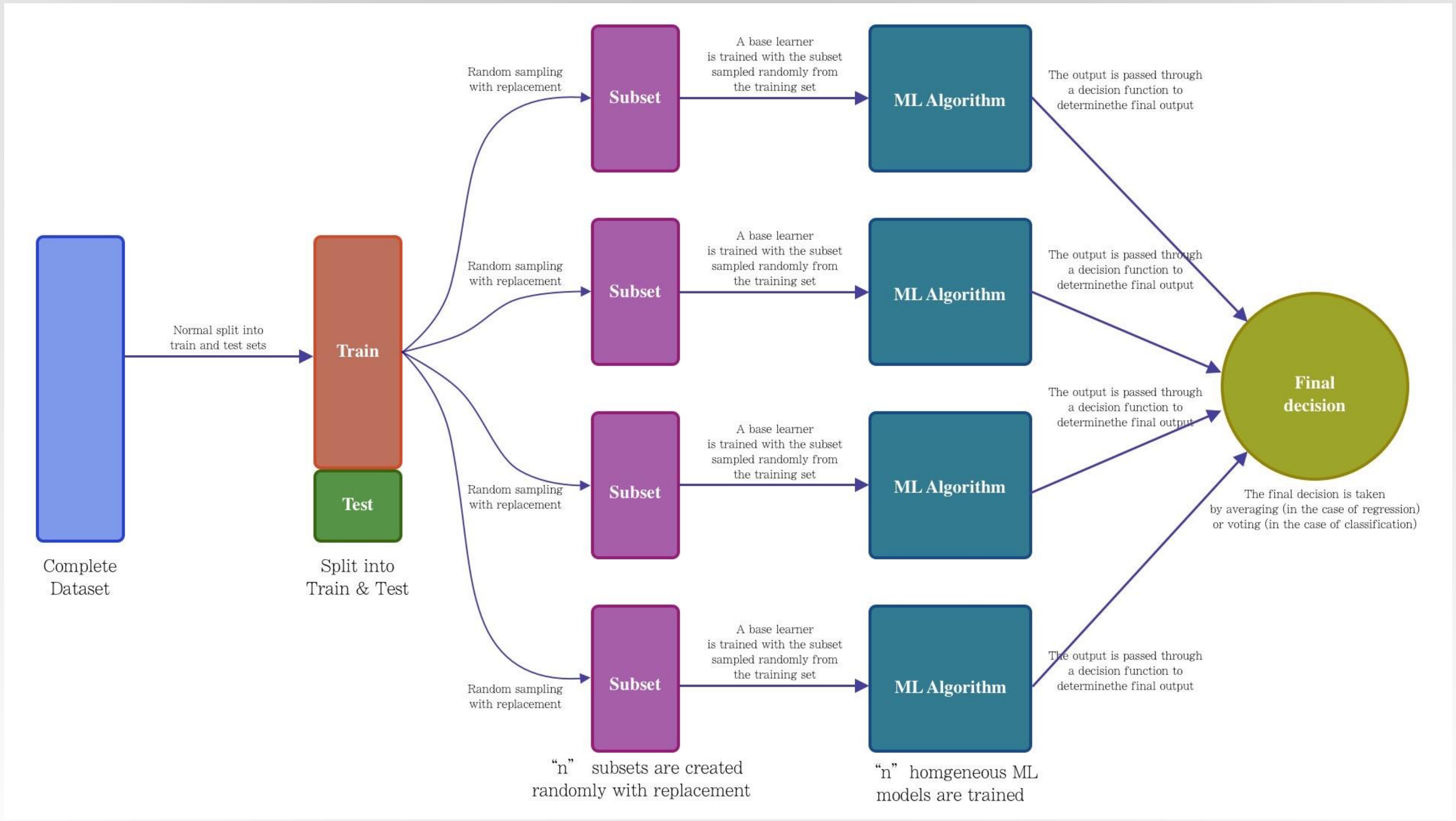


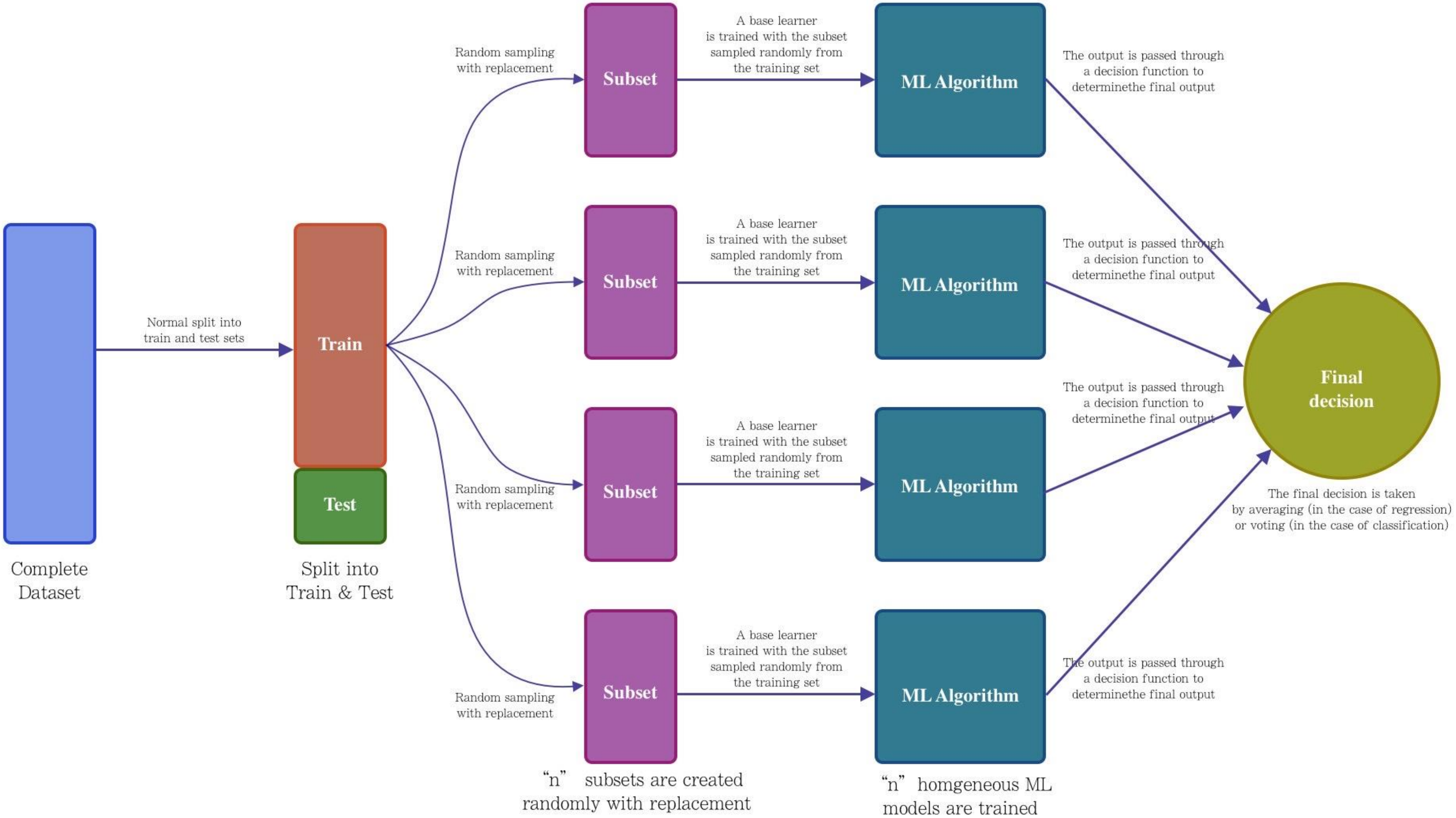
Bagging

1. Bagging consists of **creating many "copies"** of the training data
   - Each copy is slightly different from another
2. **Apply the weak learner** to each copy to obtain multiple weak models
3. **Combine** them



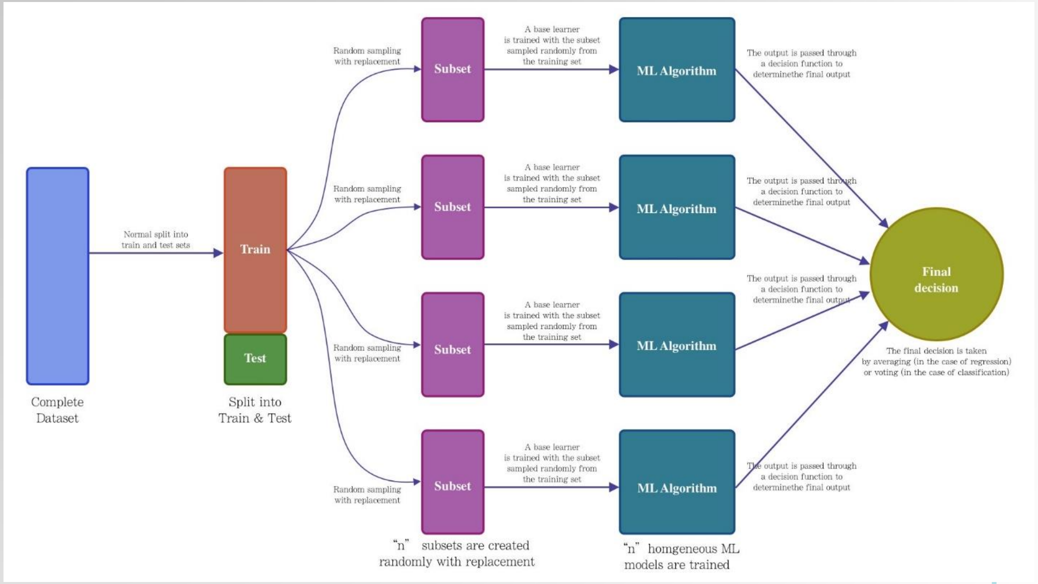
Random Forest


Complete Dataset
Normal split into train and test sets
Train
Test
Split into Train & Test
Random sampling with replacement
Subset
A base learner is trained with the subset sampled randomly from the training set
ML Algorithm
The output is passed through a decision function to determinethe final output
Final decision
The final decision is taken by averaging (in the case of regression) or voting (in the case of classification)
"n" subsets are created randomly with replacement
"n" homgeneous ML models are trained


Complete Dataset
Normal split into train and test sets
Train
Test
Split into Train & Test
Random sampling with replacement
Subset
A base learner is trained with the subset sampled randomly from the training set
ML Algorithm
The output is passed through a decision function to determinethe final output
Final decision
The final decision is taken by averaging (in the case of regression) or voting (in the case of classification)
"n" subsets are created randomly with replacement
"n" homgeneous ML models are trained

Random Forest

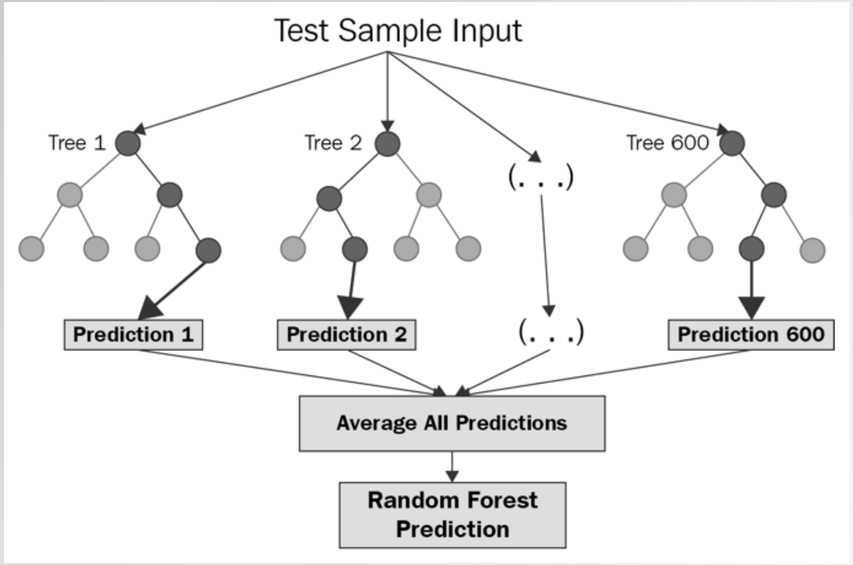
1. Given a training set, we create random samples of the training set and build a decision tree model using each sample as the training set
2. After training, we have N decision trees
3. The prediction for a new example x is obtained as the average of N predictions in the case of regression
   - Or by taking the majority vote in the case of classification


Complete Dataset
Normal split into train and test sets
Train
Test
Split into Train & Test
Random sampling with replacement
Subset
A base learner is trained with the subset sampled randomly from the training set
ML Algorithm
The output is passed through a decision function to determinethe final output
Final decision
The final decision is taken by averaging (in the case of regression) or voting (in the case of classification)
"n" subsets are created randomly with replacement
"n" homgeneous ML models are trained

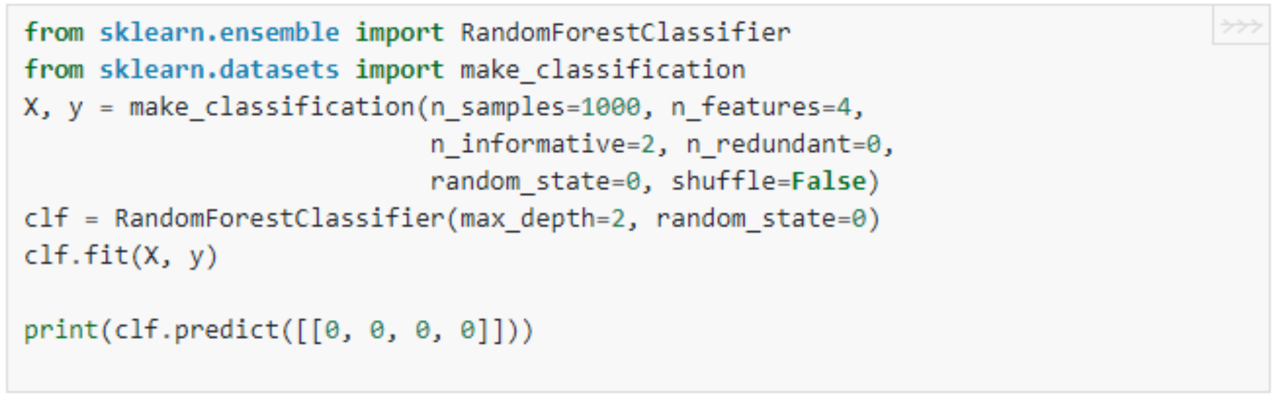
Random Forest

- The random forest algorithm is different from the vanilla bagging by using a modified tree learning algorithm that **inspects**, at each split in the learning process, a **random** subset of the features to **avoid the correlation** of the trees

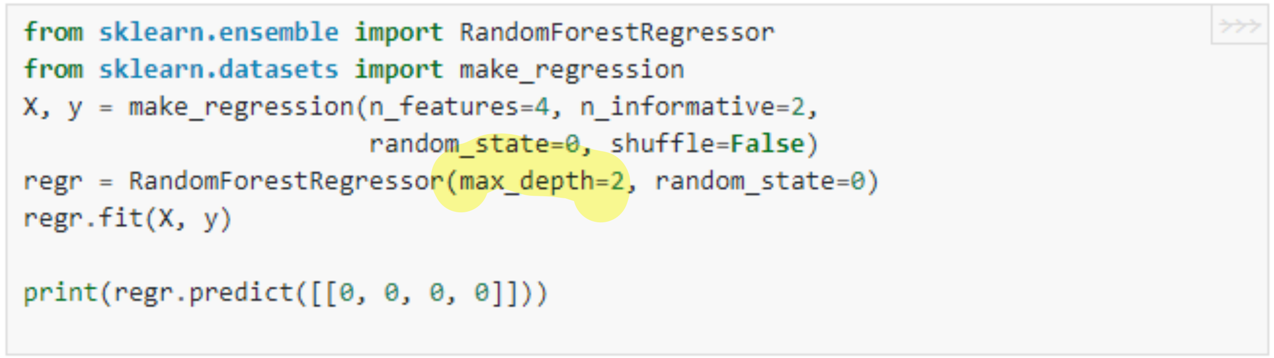
- The most important **hyperparameters** to tune
  1. Number of trees
  2. Size of the random subset of the features to consider at each split



Random Forest (scikit-learn)



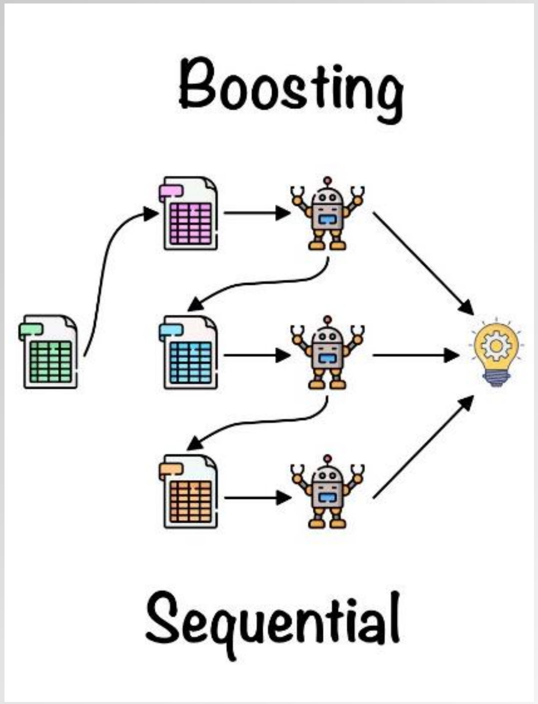
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=1000, n_features=4,
                           n_informative=2, n_redundant=0,
                           random_state=0, shuffle=False)
clf = RandomForestClassifier(max_depth=2, random_state=0)
clf.fit(X, y)

print(clf.predict([[0, 0, 0, 0]]))
```



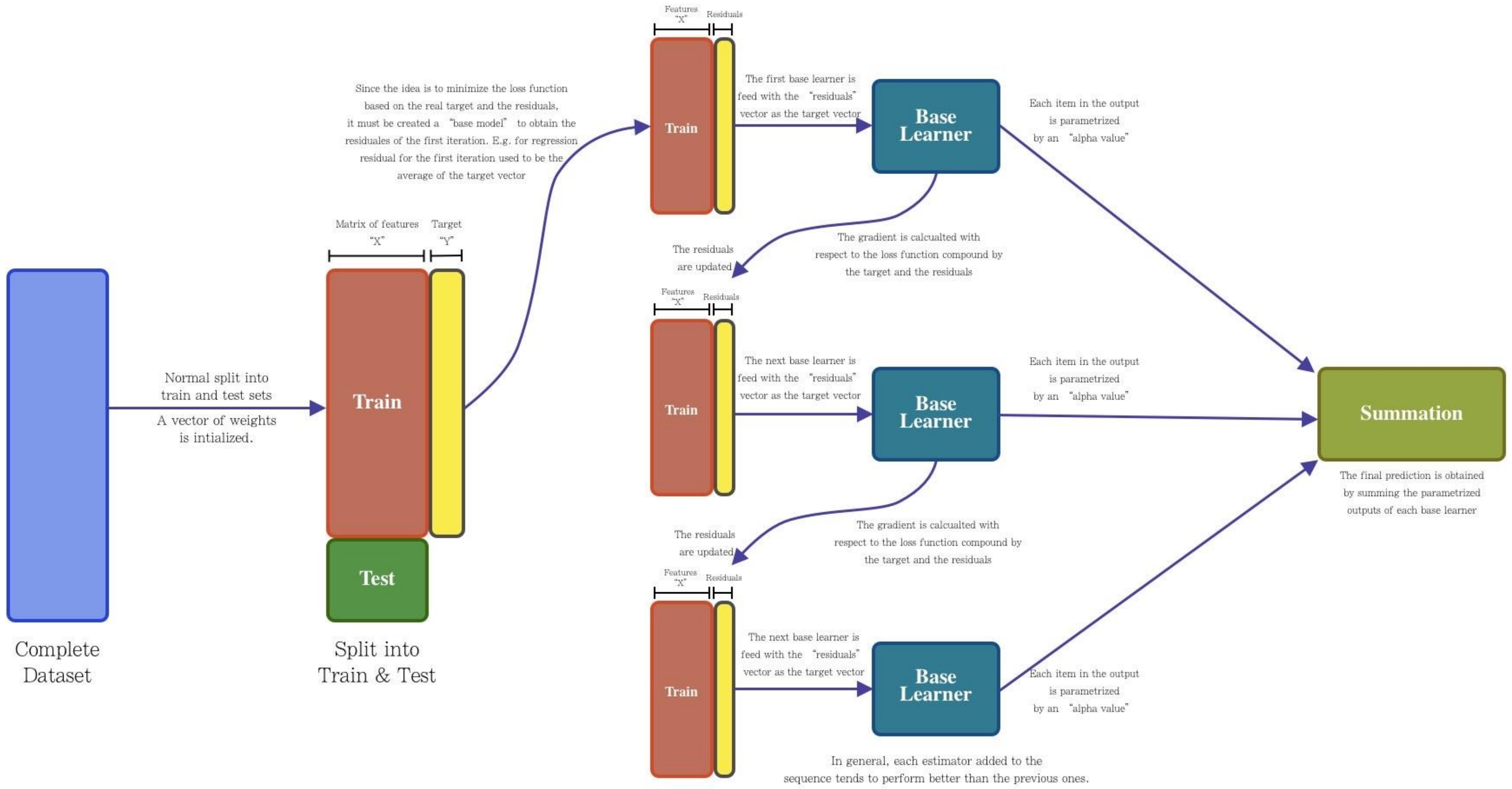
```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression
X, y = make_regression(n_features=4, n_informative=2,
                       random_state=0, shuffle=False)
regr = RandomForestRegressor(max_depth=2, random_state=0)
regr.fit(X, y)

print(regr.predict([[0, 0, 0, 0]]))
```

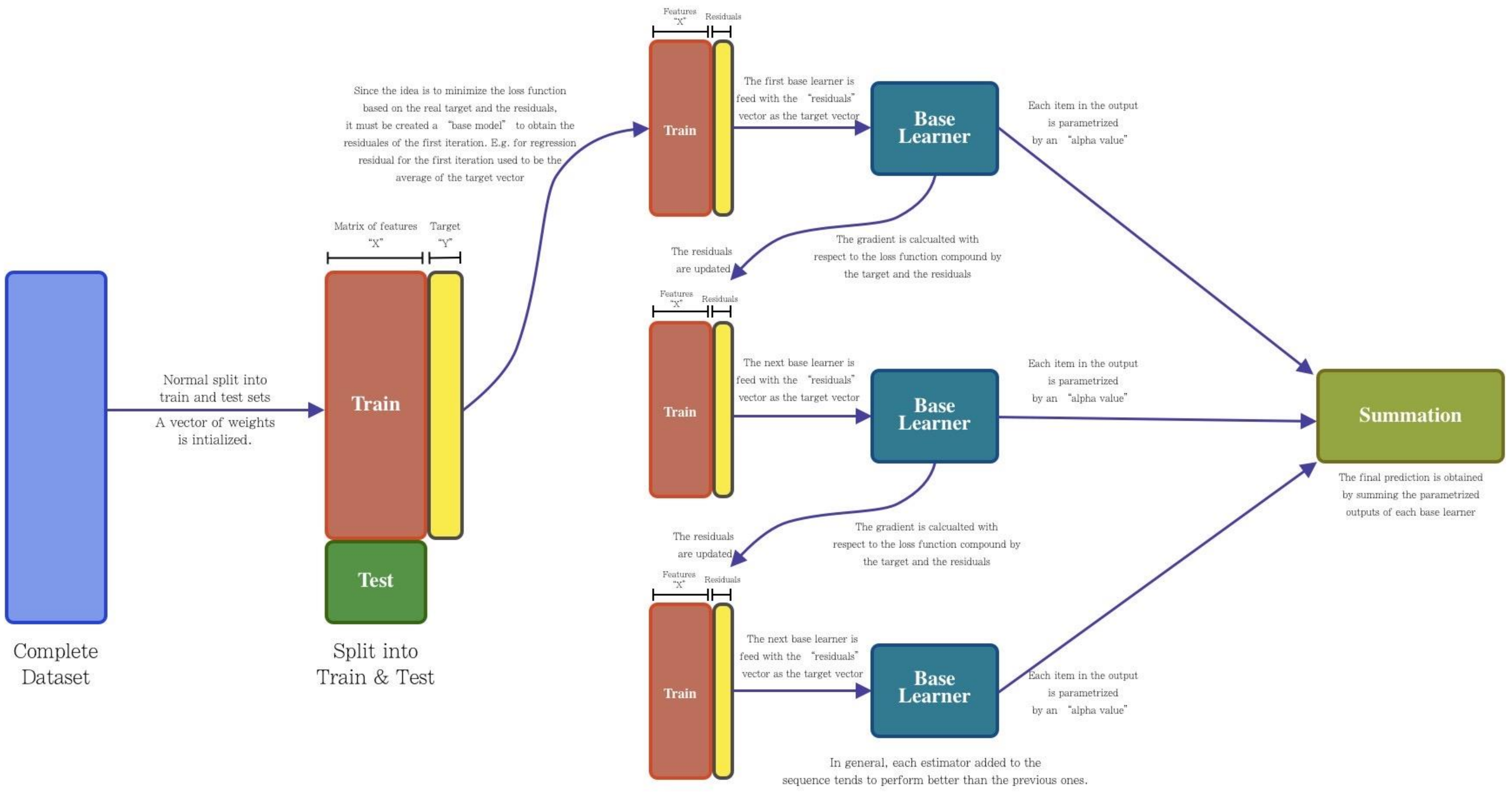
Boosting

- Boosting trains many models in a **gradual**, **additive** and **sequential** manner



Gradient Boosting


Complete Dataset
Normal split into train and test sets
A vector of weights is intialized.
Matrix of features "X"
Target "Y"
Train
Test
Split into Train & Test
Since the idea is to minimize the loss function based on the real target and the residuals, it must be created a "base model" to obtain the residuales of the first iteration. E.g. for regression residual for the first iteration used to be the average of the target vector
Features "X"
Residuals
Train
The first base learner is feed with the "residuals" vector as the target vector
Base Learner
Each item in the output is parametrized by an "alpha value"
The gradient is calcualted with respect to the loss function compound by the target and the residuals
The residuals are updated
Features "X"
Residuals
Train
The next base learner is feed with the "residuals" vector as the target vector
Base Learner
Each item in the output is parametrized by an "alpha value"
Summation
The final prediction is obtained by summing the parametrized outputs of each base learner
The gradient is calcualted with respect to the loss function compound by the target and the residuals
The residuals are updated
Features "X"
Residuals
Train
The next base learner is feed with the "residuals" vector as the target vector
Base Learner
Each item in the output is parametrized by an "alpha value"
In general, each estimator added to the sequence tends to perform better than the previous ones.


Complete Dataset
Normal split into train and test sets
A vector of weights is intialized.
Matrix of features "X"
Target "Y"
Train
Test
Split into Train & Test
Since the idea is to minimize the loss function based on the real target and the residuals, it must be created a "base model" to obtain the residuales of the first iteration. E.g. for regression residual for the first iteration used to be the average of the target vector
Features "X"
Residuals
Train
The first base learner is feed with the "residuals" vector as the target vector
Base Learner
Each item in the output is parametrized by an "alpha value"
The gradient is calcualted with respect to the loss function compound by the target and the residuals
The residuals are updated
Features "X"
Residuals
Train
The next base learner is feed with the "residuals" vector as the target vector
Base Learner
Each item in the output is parametrized by an "alpha value"
The gradient is calcualted with respect to the loss function compound by the target and the residuals
The residuals are updated
Features "X"
Residuals
Train
The next base learner is feed with the "residuals" vector as the target vector
Base Learner
Each item in the output is parametrized by an "alpha value"
In general, each estimator added to the sequence tends to perform better than the previous ones.
Summation
The final prediction is obtained by summing the parametrized outputs of each base learner

Gradient Boosting

1. Start with a constant model:

$$f = f_0 = \frac{\sum_{i=1}^{N} y_i}{N}$$

2. Modify labels of each example in our training set like follows:

$$\hat{y}_i \leftarrow y_i - f(x_i)$$

   - where $\hat{y}_i$, called the ***residual***, is the new label for example x_i

3. Use the modified training set, with residuals instead of original labels, to build a new decision tree model, f_1

- The boosting model is now defined as

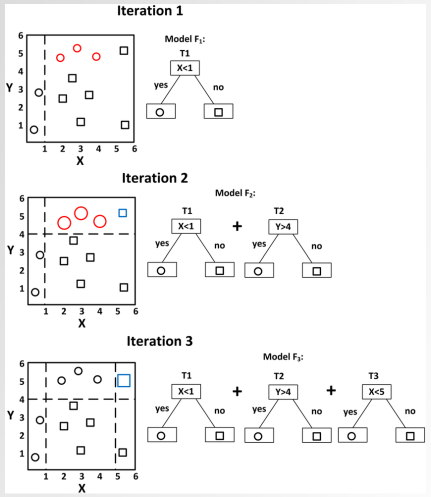
$$f = f_0 + \alpha f_1$$

   - where $\boldsymbol{\alpha}$ is the learning rate (a hyperparameter)

4. Recompute the residuals using equation in step 2 and replace the labels in the training data once again, train the new decision tree model f_2, redefine the boosting model as

$$f = f_0 + \alpha f_1 + \alpha f_2$$

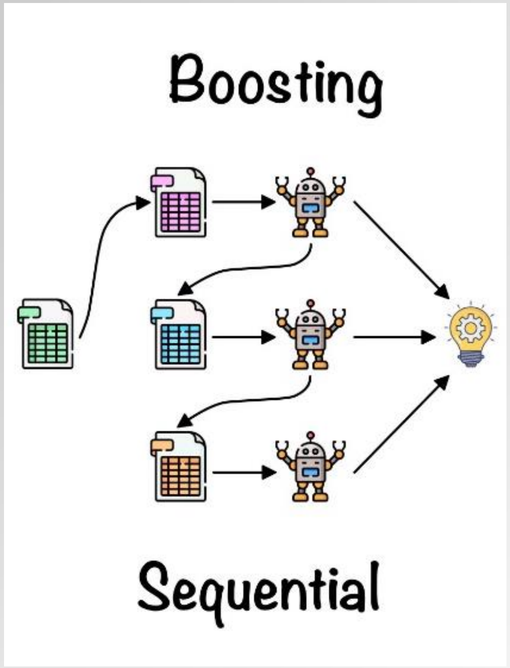
- And the process continues until the maximum of **M** (another hyperparameter) trees are combined



Gradient Boosting

- **Hyperparameters** to tune in gradient boosting:
    1. Number of trees
    2. Learning rate (α)
    3. Depth of trees

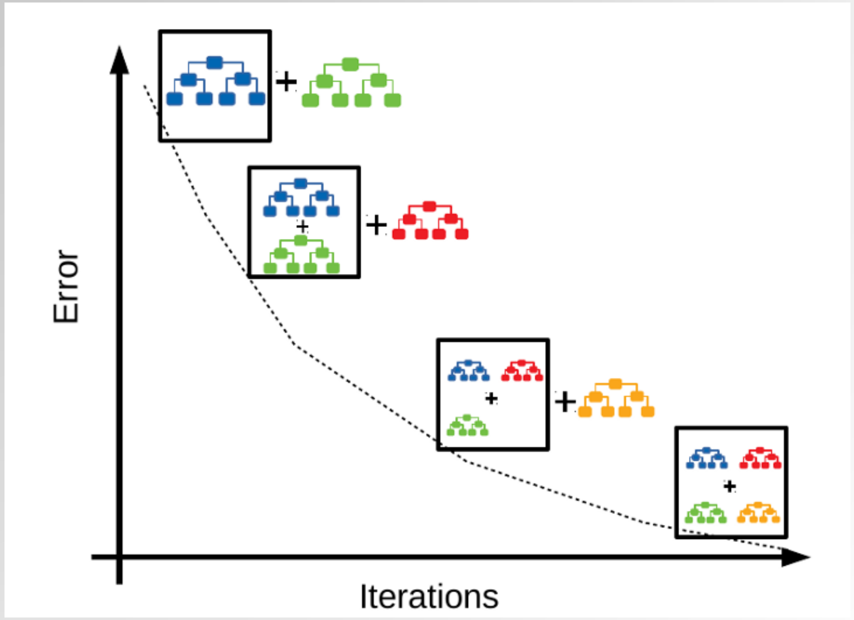
- All the hyperparameters affect model accuracy
- The depth of trees also affects the speed of training and prediction: the shorter, the faster



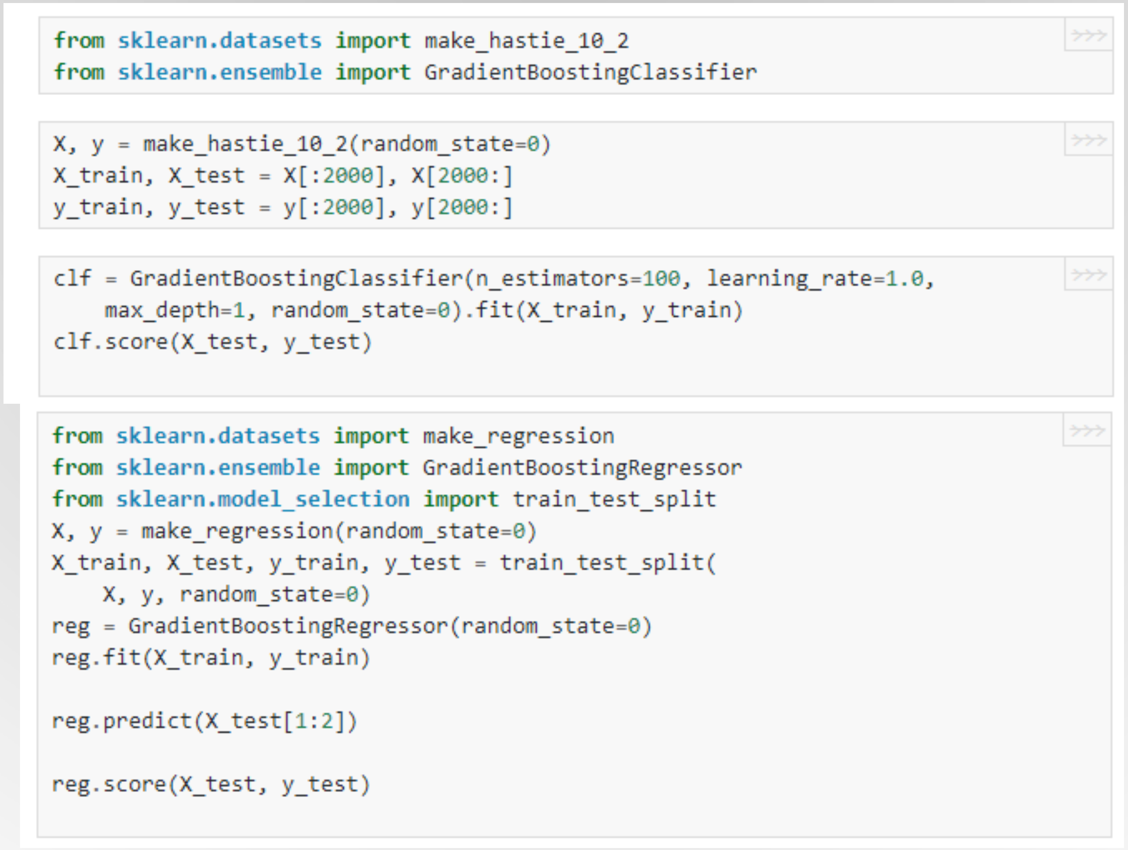
Gradient Boosting

- Gradient boosting is one of the most powerful machines learning algorithms
  - It creates very **accurate** models
  - It is capable of **handling huge datasets** with millions of examples and features

- It usually outperforms random forest in accuracy but can be significantly **slower** in training because of its sequential nature



Gradient Boosting (scikit-learn)



```
from sklearn.datasets import make_hastie_10_2
from sklearn.ensemble import GradientBoostingClassifier
```

```
X, y = make_hastie_10_2(random_state=0)
X_train, X_test = X[:2000], X[2000:]
y_train, y_test = y[:2000], y[2000:]
```

```
clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
    max_depth=1, random_state=0).fit(X_train, y_train)
clf.score(X_test, y_test)
```

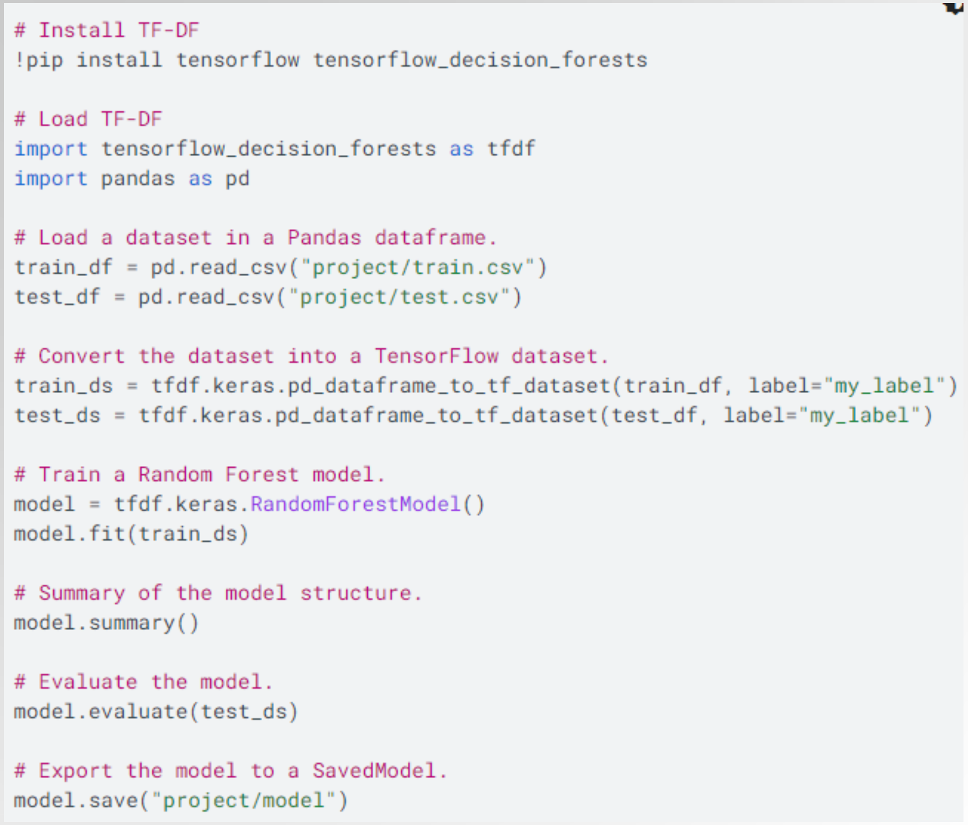
```
from sklearn.datasets import make_regression
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split
X, y = make_regression(random_state=0)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, random_state=0)
reg = GradientBoostingRegressor(random_state=0)
reg.fit(X_train, y_train)

reg.predict(X_test[1:2])

reg.score(X_test, y_test)
```

TensorFlow Decision Forests (TF-DF)

- TensorFlow Decision Forests (TF-DF) is a library for the training, evaluation, interpretation and inference of Decision Forest models.
- The two most popular DF training algorithms are Random Forests and Gradient Boosted Decision Trees.

TensorFlow Decision Forests (TF-DF)

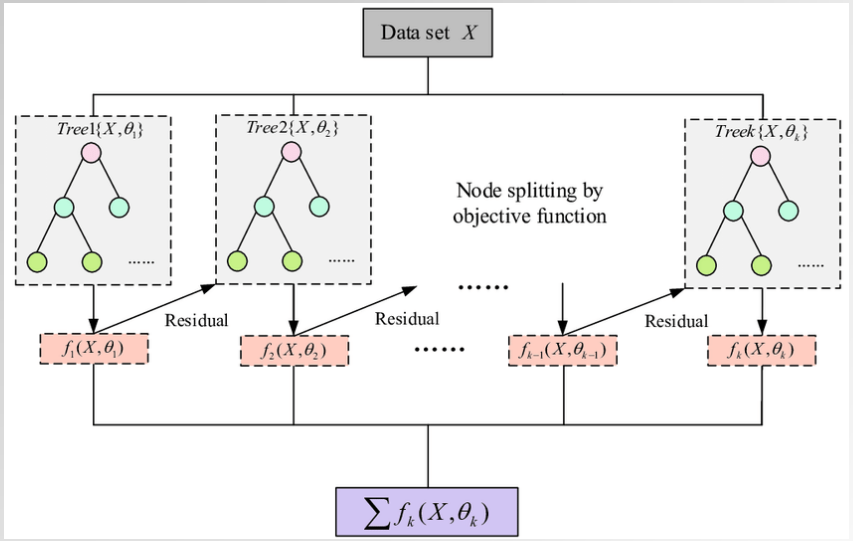
https://colab.research.google.com/github/tensorflow/decision-forests/blob/main/documentation/tutorials/beginner_colab.ipynb



```
# Install TF-DF
!pip install tensorflow tensorflow_decision_forests

# Load TF-DF
import tensorflow_decision_forests as tfdf
import pandas as pd

# Load a dataset in a Pandas dataframe.
train_df = pd.read_csv("project/train.csv")
test_df = pd.read_csv("project/test.csv")

# Convert the dataset into a TensorFlow dataset.
train_ds = tfdf.keras.pd_dataframe_to_tf_dataset(train_df, label="my_label")
test_ds = tfdf.keras.pd_dataframe_to_tf_dataset(test_df, label="my_label")

# Train a Random Forest model.
model = tfdf.keras.RandomForestModel()
model.fit(train_ds)

# Summary of the model structure.
model.summary()

# Evaluate the model.
model.evaluate(test_ds)

# Export the model to a SavedModel.
model.save("project/model")
```

XGBoost

- XGBoost, which stands for Extreme Gradient Boosting, is a scalable, distributed gradient-boosted decision tree machine learning library.
- It provides **parallel tree boosting** and is the **leading** machine learning library for regression, classification, and ranking problems.



XGBoost

```
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# load data
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=",")
# split data into X and y
X = dataset[:,0:8]
Y = dataset[:,8]
# split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, random_state=42)
# fit model no training data
model = XGBClassifier()
model.fit(X_train, y_train)
# make predictions for test data
y_pred = model.predict(X_test)
#Predictions made by XGBoost are probabilities. Convert them to binary class values by rounding them to 0 or 1.
predictions = [round(value) for value in y_pred]
# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```