CS5805 : Machine Learning I
Lecture # 16

Reza Jafari

Collegiate Associate Professor
rjafari@vt.edu

Ensemble Learning background

- Ensemble-based decision making has been in round in our daily lives for many years. As long as civilized communities existed.
- We often seek the opinions of different experts to help us make the decision.
- Unity is strength old saying underlying the powerful ensemble learning in machine learning.
- Here are example of ensemble-based decision making:
  - Essence of democracy where group of people vote.
  - Judicial system and jury of peers, a panel of judges.
  - Reading reviews before purchasing an item.
  - Consulting with several doctors prior to major medical operation.
  - Peer review of an article prior to publication.
- Ensemble learning: Improve our confidence of making right decision by weighing various opinions and combining them through some thought process to reach final decision.

Weak Learner versus Strong Learner

Weak Learner (WL)

A weak learner is a learning algorithm capable of producing simple classifiers with probability error strictly less than that of random guessing (0.5 in the binary case).

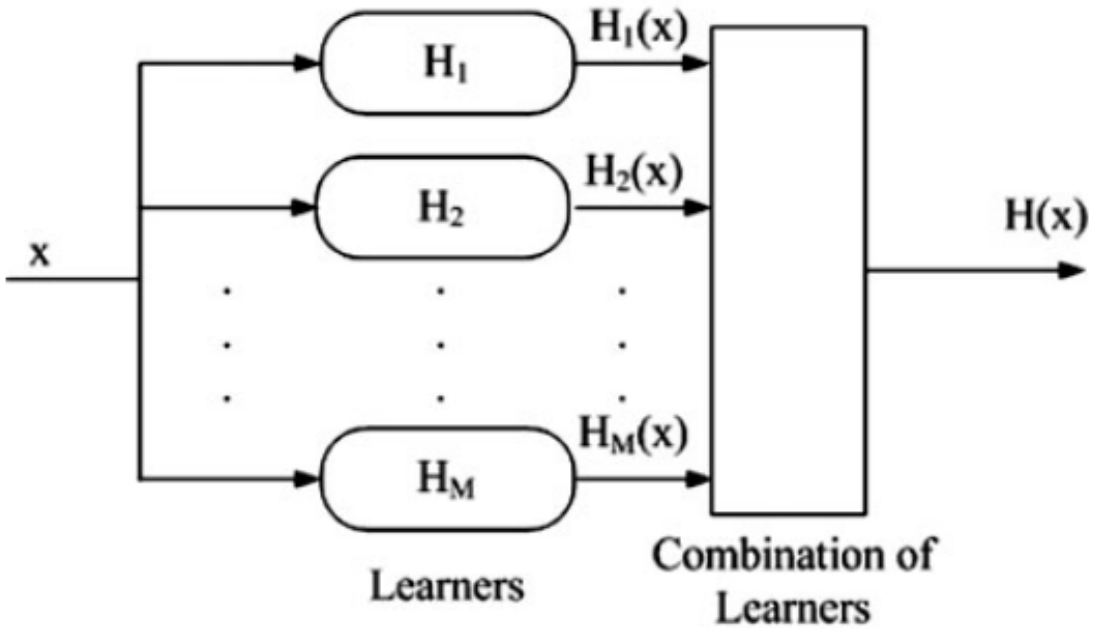
Strong Learner(SL)

A strong learner is able (given enough training data) to yield classifiers with arbitrary small error probability.

- An ensemble (or committee) of classifiers is a classifier build upon some combination of WLs.
- Ensemble classifiers is to learn many *weak classifiers* and combine them in some way, instead of trying to learn a single classifier.
- i.e. large neural network, we may train several NNs and combine individual outputs to produce final outputs.

Combination of Learners

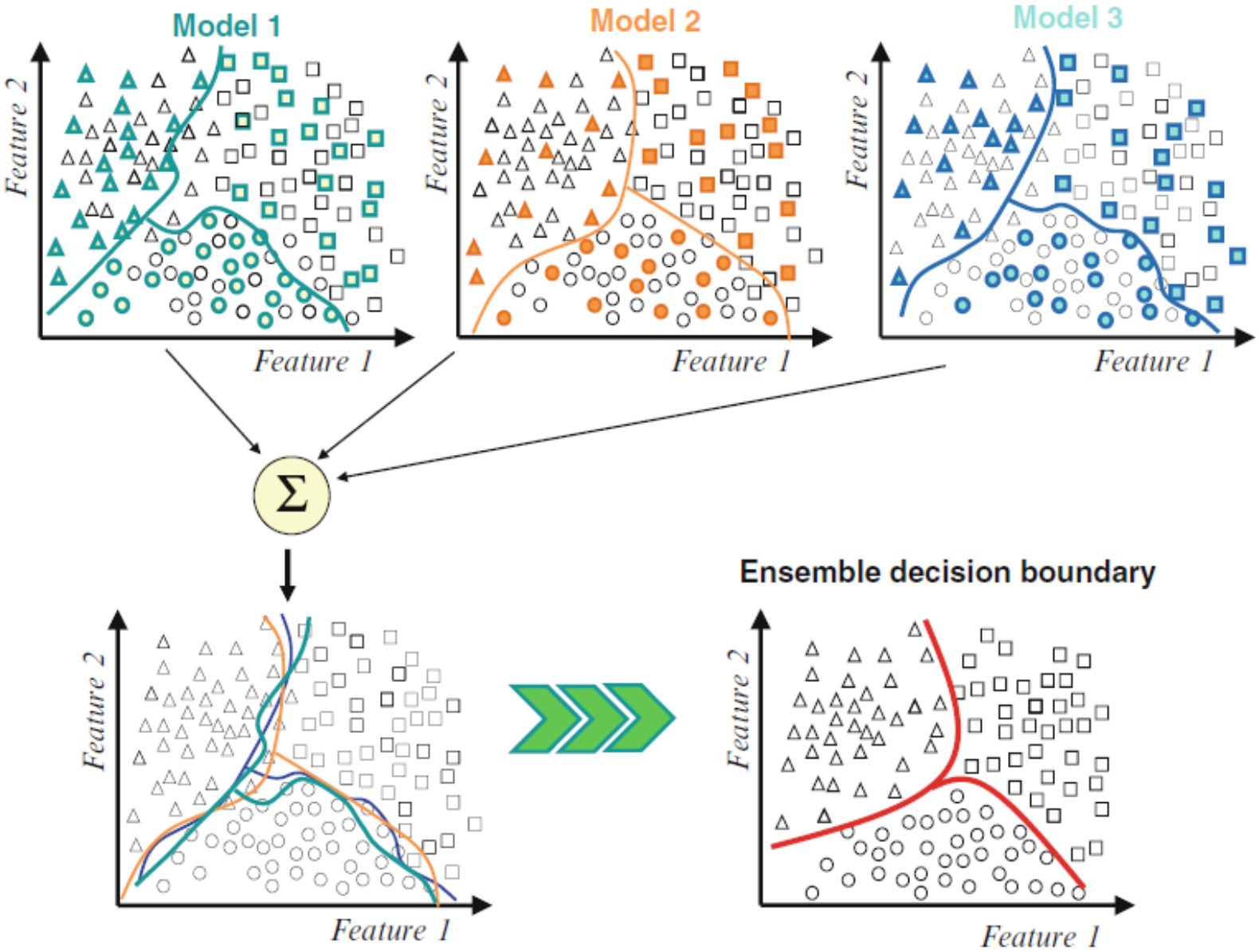
- The outputs of weak learners $H_m(x)$ with $m \in \{1, ..., M\}$ are combined to produce the output of the ensemble classifiers by $H(x)$



Statistical and computational Justification for Ensemble systems

- Any classification error is composed of two components:
  - bias : the accuracy of the classifier
  - variance: the precision of the classifier when trained on different training set.
- Often bias-variance have a trade-off relationship: classifiers with low bias tend to have high variance and vice versa.
- The goal of ensemble systems is to create several classifiers with relatively fixed bias and then combining their outputs, i.e. averaging, to reduce variance.
- Two issues:
  - There are many ways of combining ensemble members. Averaging is only one method.
  - Combining the classifier outputs not necessarily lead to a classification performance.

Variability reduction using ensemble systems



Bias-variance decomposition

- Let $x_1, ..., x_n$ be a training set and y_i a real value associated with x_i.
- We assume there is a real function with noise:

$$y = f(x) + \epsilon$$

where ϵ has a zero mean and variance of σ^2.

- We want to find a function $\hat{f}((x); D)$ that approximate $f(x)$ using learning algorithms on a training dataset :

$$D = \{(x_1, y_1, ..., (x_n, y_n)\}$$

- The expected value of the Mean square error between y and $\hat{f}(x; D)$ measure the precision of the prediction.

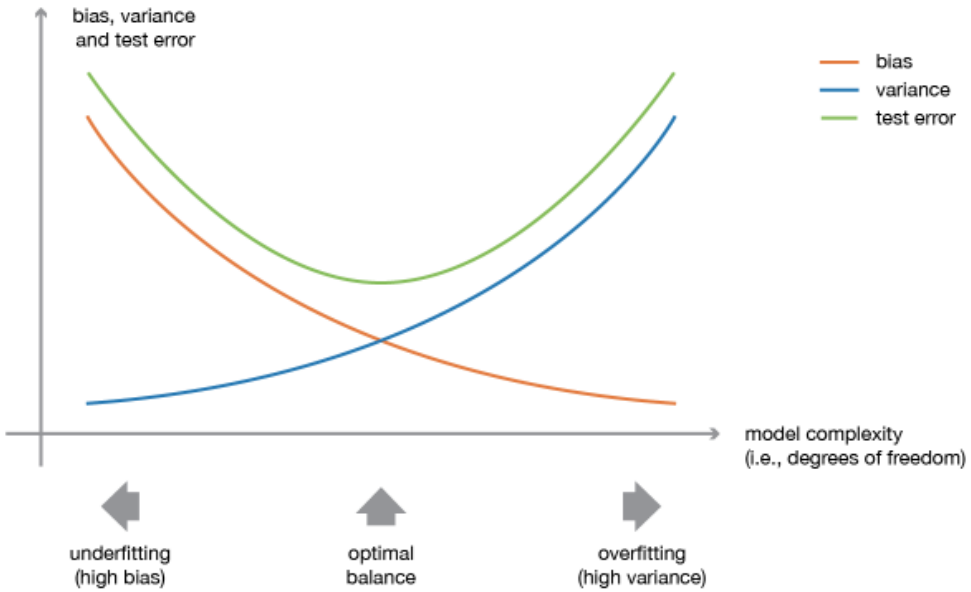
$$E_{D,\epsilon}[(y - \hat{f}(x; D))^2] = (Bias_D[\hat{f}(x; D)])^2 + var_D[\hat{f}(x; D)] + \sigma^2$$

Bias-variance decomposition

- $Bias_D^2$ of the predictive model: the error by simplifying built into the model.
- Example: Approximating a non-linear function $f(x)$ using a linear model.
- Var_D: how much the predictive model $\hat{f}(x)$ will move around its mean.
- σ^2: irreducible error.
- All three terms are non-negative, so the σ^2 forms a lower bound on the expected error on unseen samples.

Variance-bias dilemma

- A low bias and low variance : two most fundamental features expected for a model.
- Often bias-variance have a trade-off relationship: classifiers with low bias tend to have high variance and vice versa.
- Model to have enough degree of freedom to resolve the underlying complexity but not too much degree of freedom to avoid high variance (be robust).
- The goal of ensemble systems are to reduce variance.



Ensemble Learning

- Ensemble learning refers to algorithms that combine the predictions from two or more models.
- Theoretically unlimited number of ensembles can be developed for predictive modeling.
- Three pillars of the ensemble systems:1- diversity, 2- training ensemble members, 3- combining ensemble systems.
- The three main classes of ensemble learning methods are:

Bagging

Stacking

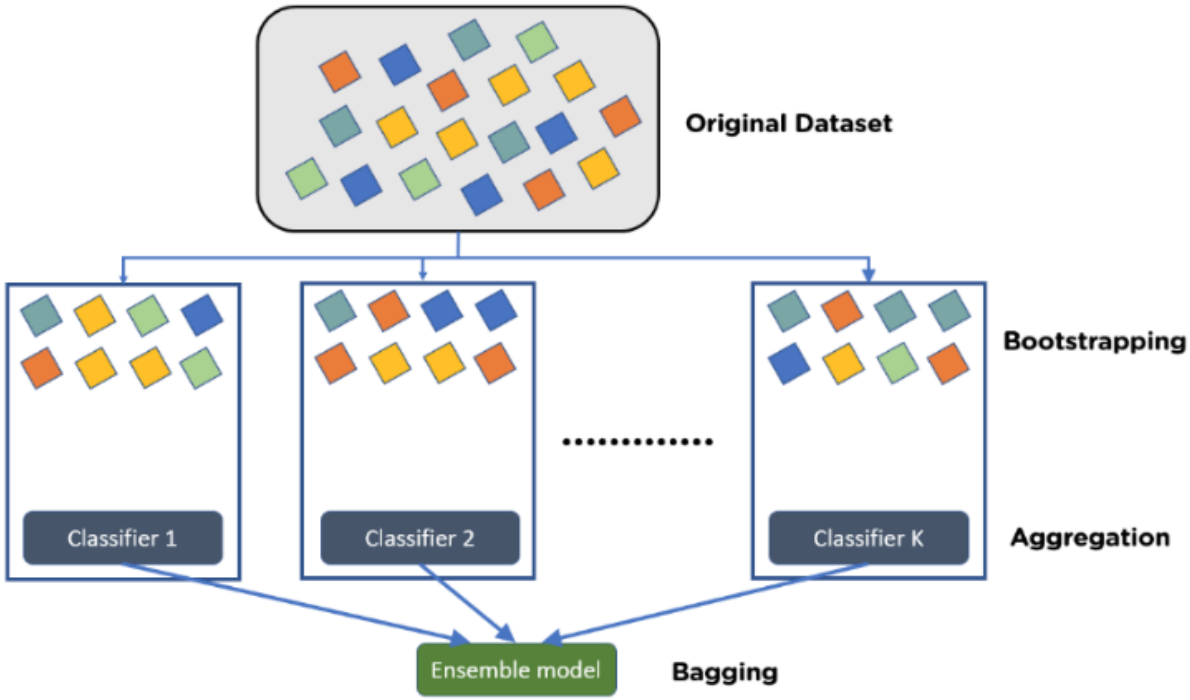
Boosting

Bagging Ensemble Learning

- Bootstrap aggregation or in short bagging, is an ensemble learning method that seeks a diverse group of ensemble members by varying the training data with replacement.
- The two key ingredients of bagging are bootstrap and aggregation.
- Decision trees are simple predictive modeling technique, but they suffer from high-variance. Very different results given different training data.
- Bagging is a technique to make decision tree more robust and achieve better performance.
- Training each model on a different sample of the same training dataset.
- The predictions made by the ensemble members are then combined using simple statistics, such as voting or averaging.

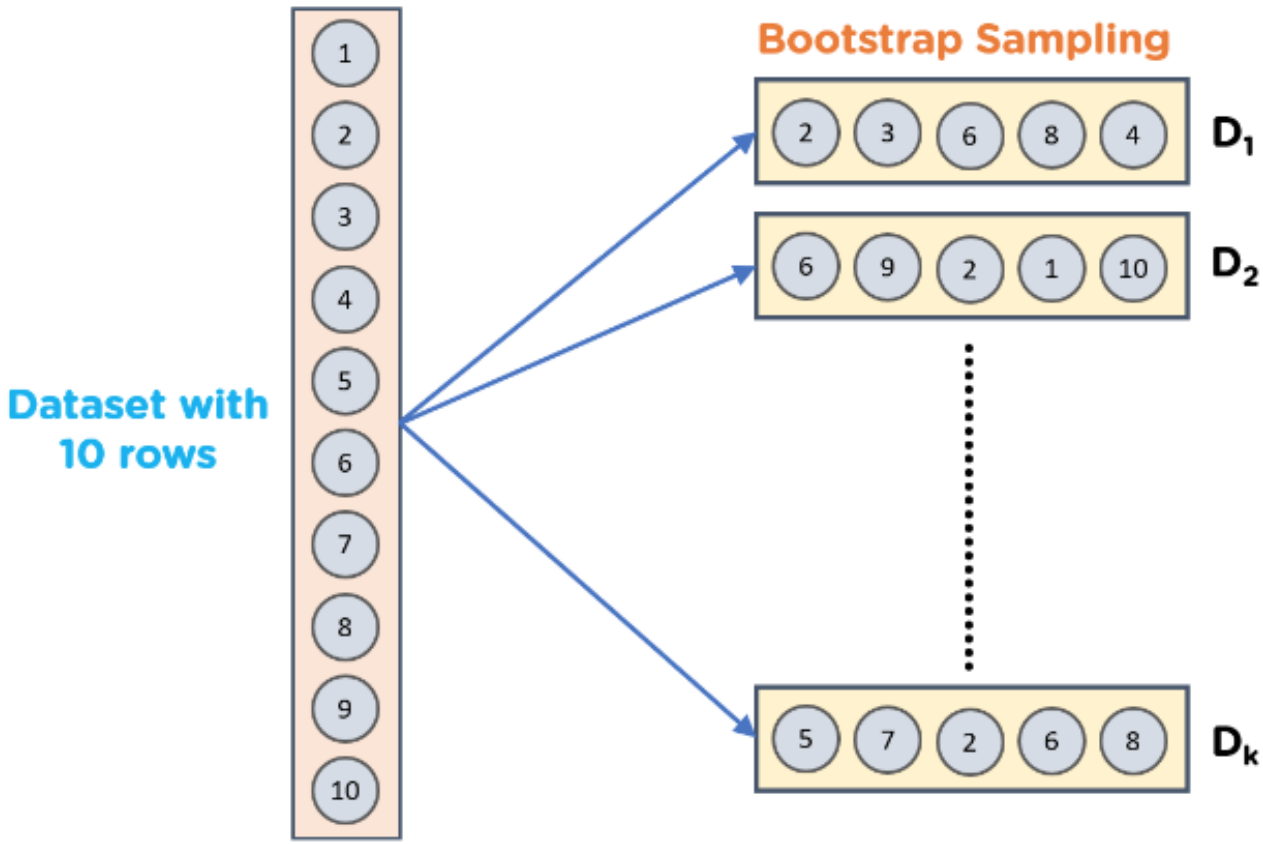
Bagging Ensemble Learning

- Bagging helps to improve the performance and accuracy of machine learning algorithms.
- It is used to reduce the variance of a prediction model.
- Avoid overfitting and it is used for classification and regression.



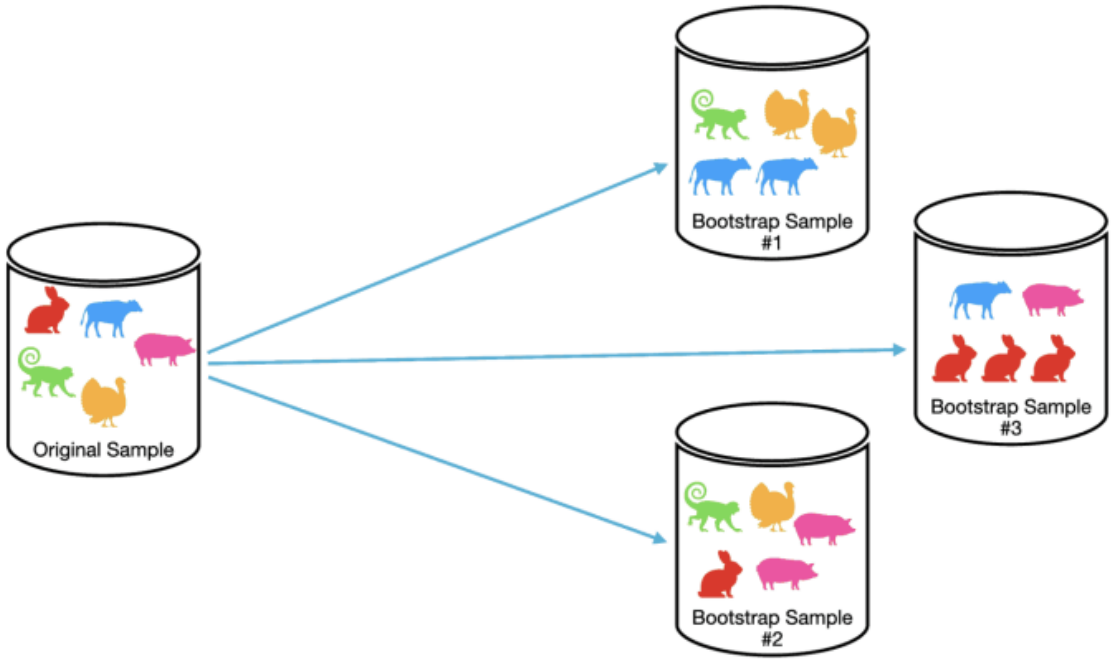
What is bagging?

- A random sample of data in a training set is selected with replacement.
- Meaning that the individual data points can be chosen more than once.



What is bagging?

1. Equal probability of selected data.(cow, pig, turkey, pig and monkey)
2. Select data point from original sample with replacement.
3. Repeat above step until the current bootstrap sample is the same size as the original sample.
4. Repeat step 2&3 until the required number of bootstrap are obtained.



Steps to perform bagging

1. Given a training dataset S of cardinality N, m features in the training set, bagging trains T independent classifiers, each trained by sampling, with replacement, N instances (or some % of N).
2. A subset of m features is chosen randomly to create a model using sample observations.
3. The feature offering the best split out of the lot is used to split the nodes.
4. The tree is grown, so you have the best root nodes.
5. The above steps are repeated n times. It aggregates the output of individual decision trees to give the best prediction.

Advantages of Bagging

- Reduce over-fitting and variance.
- Best for problem with relatively small training dataset, solves classification and regression problems.

Algorithm Bagging

- Inputs: Training data S; Supervised learning algorithm.

for $t = 1, ..., T$

1. Take a bootstrap replica S_t by randomly drawing $R\%$ of S.
2. Train **Base classifier** with S_t and receive hypothesis (classifier) h_t.
3. Add h_t to the ensemble. $\varepsilon \leftarrow \varepsilon \cup h_t$.

Ensemble Combination: Simple Majority Voting

1. Evaluate the ensemble $\varepsilon = \{h_1, ..., h_T\}$ on unlabeled instance x.
2. Let $v_{t,c} = 1$ if h_t chooses class ω_c and 0, otherwise.
3. Obtain total vote received by each class
   $V_c = \sum_{t=1}^{T} v_{t,c}, c = 1, ..., C$

- **Output:** Class with the highest V_c.

Bagging-Python-sklearn

- Breast cancer dataset and classification using decision tree-ensemble learning.
- 569 observations and 30 features.
- 96% accuracy.
- sklearn package in python.

Steps for typical machine learning problem

1. Import libraries.
2. Perform Explanatory Data Analysis (EDA)
3. Divide data into training and testing sets, i.e. 70-30%
4. Pick a classifier and train the algorithm.
5. Make a prediction.
6. Evaluate the algorithm's performance.

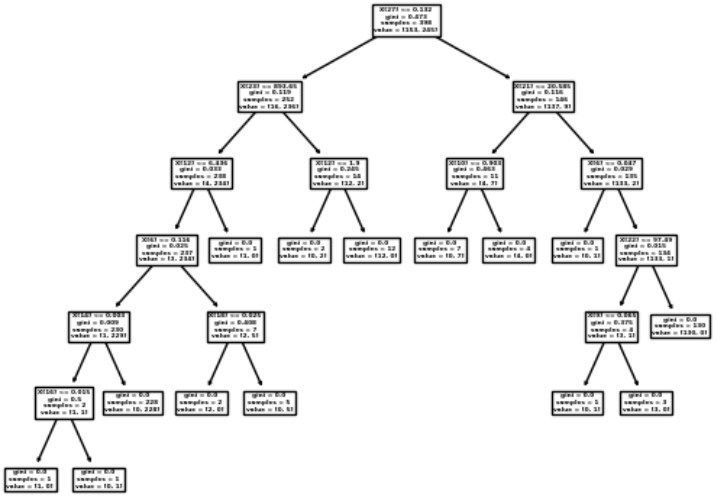
Sample code - Bagging

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn import tree

data = load_breast_cancer()

X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=17)

bdtc = BaggedDecisonTreeClassifier(len(X_train))
# Fitting the model
bdtc.fit(X_train,y_train)
# Predicting with model
y_pred = bdtc.predict(X_test)
# Calculating Accuracy
print(bdtc.acc(y_test,y_pred))
```

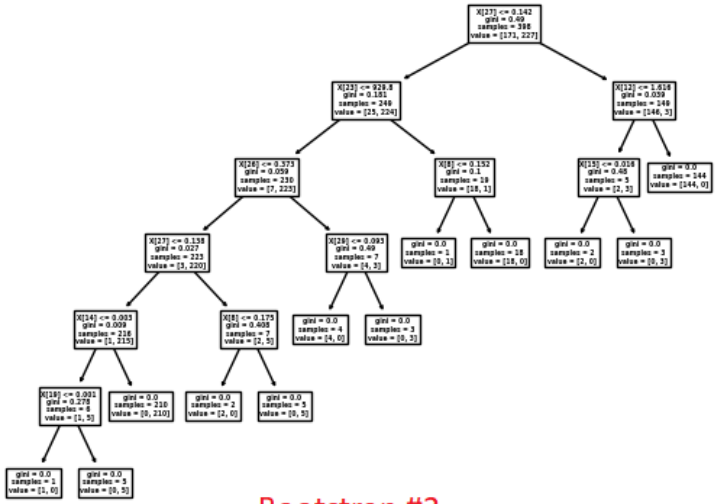
Sample code - Bagging

```
class BaggedDecisonTreeClassifier:

    def __init__(self, num_of_bagged=5):
        # Initialised with number of bagged models
        self.num_of_bagged = num_of_bagged

    def fit(self, X, y):
        # to store the models
        self.models = []
        for i in range(self.num_of_bagged):
            indexs = np.random.choice(len(X), size=len(X))  # sample with replacement
            Xi = X[indexs]  # Chossing random samples
            Yi = y[indexs]
            # Training for each sample bunch by Decision Tree Classifier
            model = DecisionTreeClassifier()
            model.fit(Xi, Yi)
            _ = tree.plot_tree(model)
            plt.show()
            # Storing the models
            self.models.append(model)

    def predict(self, X):
        pred = np.zeros(len(X))
        # predicting with each stored models
        for model in self.models:
            pred = pred + model.predict(X)
        return np.round(pred / self.num_of_bagged)  # Model averaging

    def acc(self, y_true, y_pred):
        return np.mean(y_true == y_pred)
```

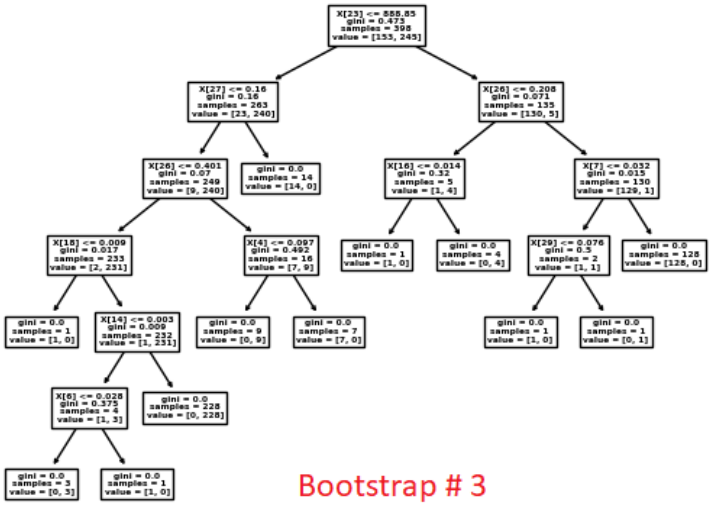
Bootstrap samples & decision tree



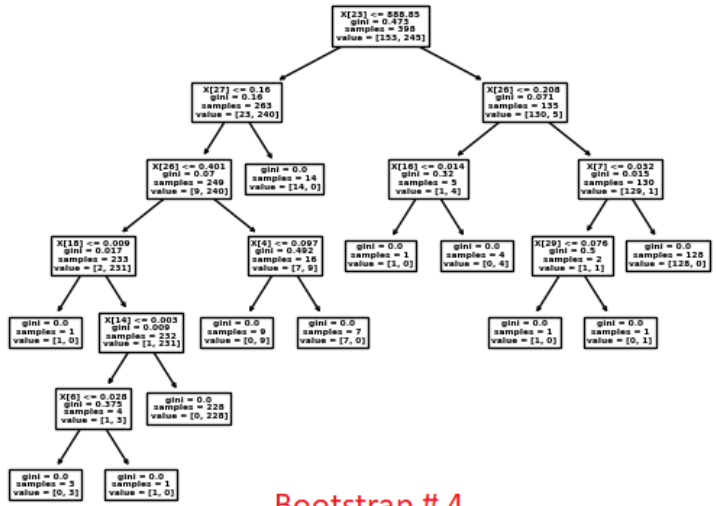
Bootstrap # 1



Bootstrap #2



Bootstrap # 3

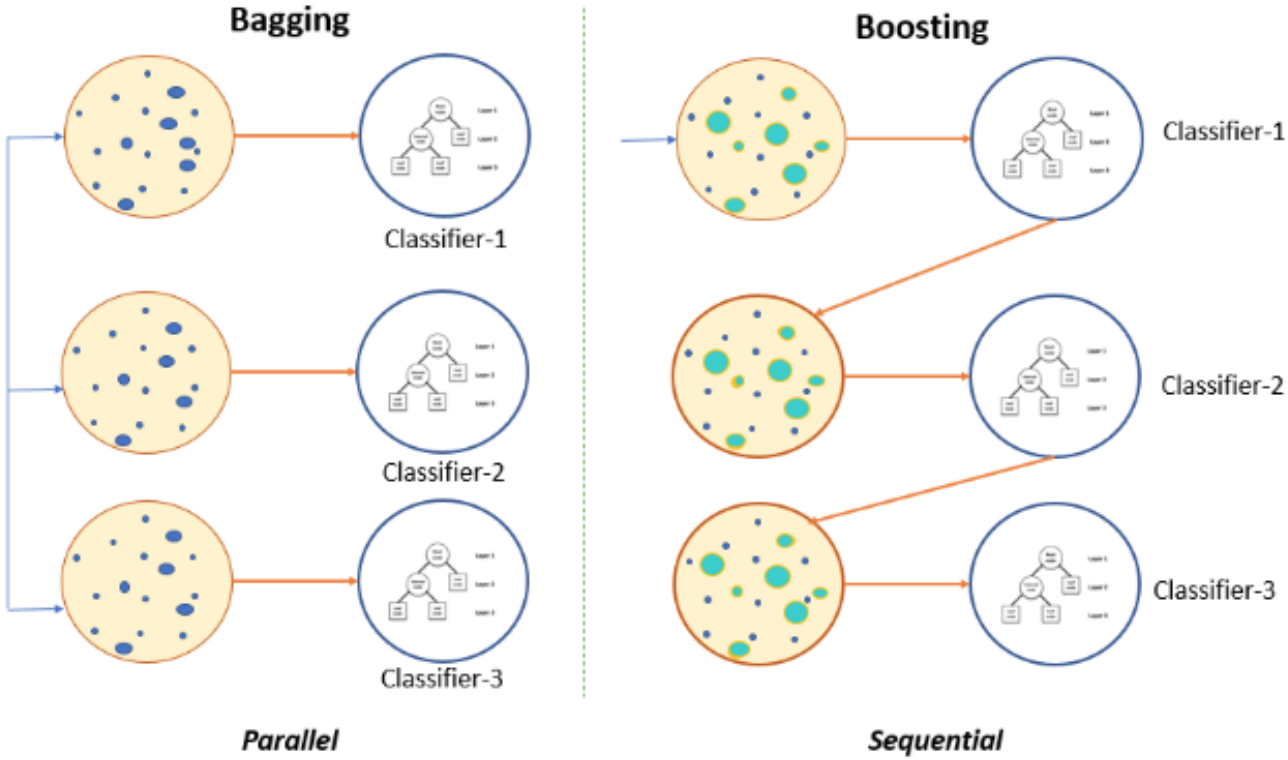


Bootstrap # 4

....

Boosting

- In **bagging**, multiple homogenous algorithms are trained **independently** in **parallel**.
- While in Boosting multiple homogenous algorithms are trained sequentially.



Bagging versus Boosting

- Bagging and Boosting both use an arbitrary number of learners by generating additional data while training.
- Bagging : allows multiple similar models (trained independently in parallel) with high variance are averaged to decrease variance.
- Boosting : builds multiple incremental models (trained sequentially focused on misclassified samples) to decrease bias, while keeping the variance small.

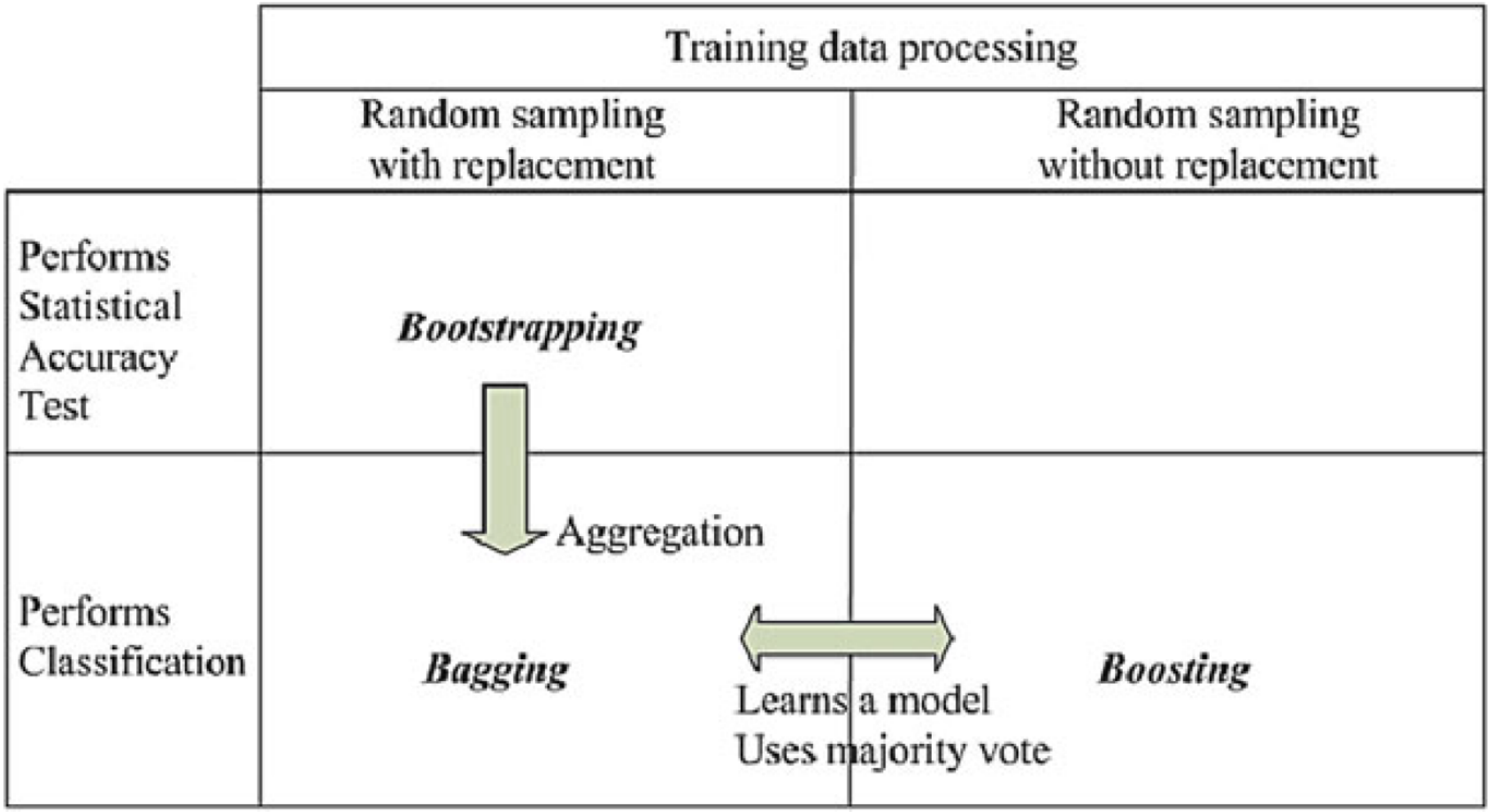
Adaptive boosting (AdaBoost)

eXtreme Gradient Boosting (XGBoost)

Light Gradient Boosting Method (LightGBM)

Category Boosting(CatBoost)

Bagging versus Boosting

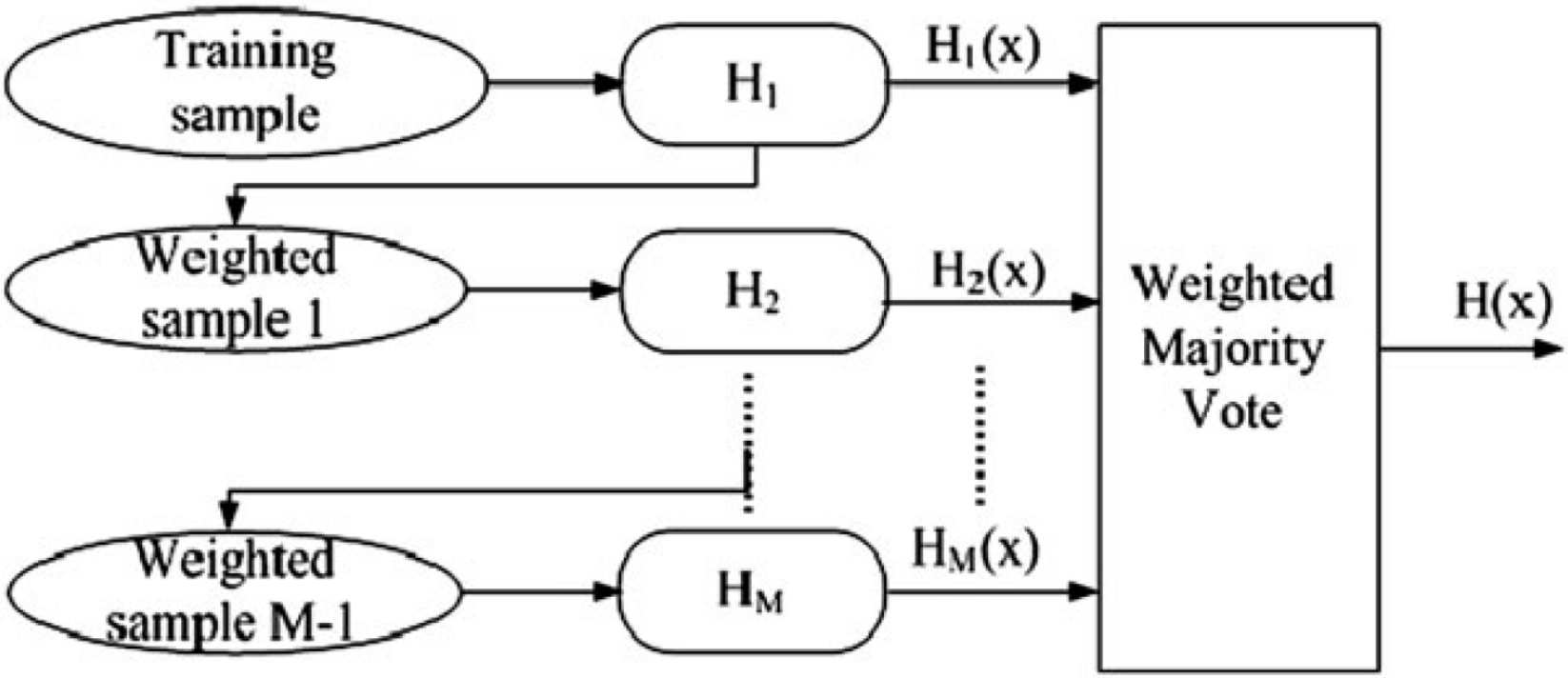


AdaBoost

- AdaBoost or adaptive boosting focuses on enhancing the performance in area where the base learners fails.
- A base learner is the first iteration of the model.
- The algorithm learns a set of weak learner classifiers sequentially using re-weighted version of the training data.
- The training set is always the same at each iteration, with each training instance weighted according to misclassification by the previous classifiers.
- This allows the WL at each iteration to focus on patterns that were not classified by the previous weak classifiers.
- It is important to chose WLs to obtain the base classifiers.
- If the base learner is too strong, it may achieve high accuracy, leaving only outliers and noisy instances with significant weight to be learned in the next iterations.

Adaptive Boosting

- Each weak learner is trained on a different weighted version of the training data.
- The weight of each instance for the following round depends on the perfromance of the previous round.



AdaBoosting for classification-skelearn-python

AdaBoost for Classification

1. make_classification() function to create a synthetic binary classification dataset.
2. Define the model by calling AdaBoostClassifier() .
3. Evaluate the model using k-fold CV with 3 repeats and 10 folds.
4. Report performance.

Adaptive Boosting for classification

```python
# test classification dataset
import numpy as np
from sklearn.datasets import make_classification
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold

X, y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_redundant=5, random_state=6)
# summarize the dataset
print(X.shape, y.shape)

model = AdaBoostClassifier()
# evaluate the model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1, error_score='raise')

# report performance
print(f'Accuracy: mean : {np.mean(n_scores):.3f} (standard deviation : {np.std(n_scores):.3f})')
```

AdaBoosting for Regression-skelearn-python

AdaBoost for Regression

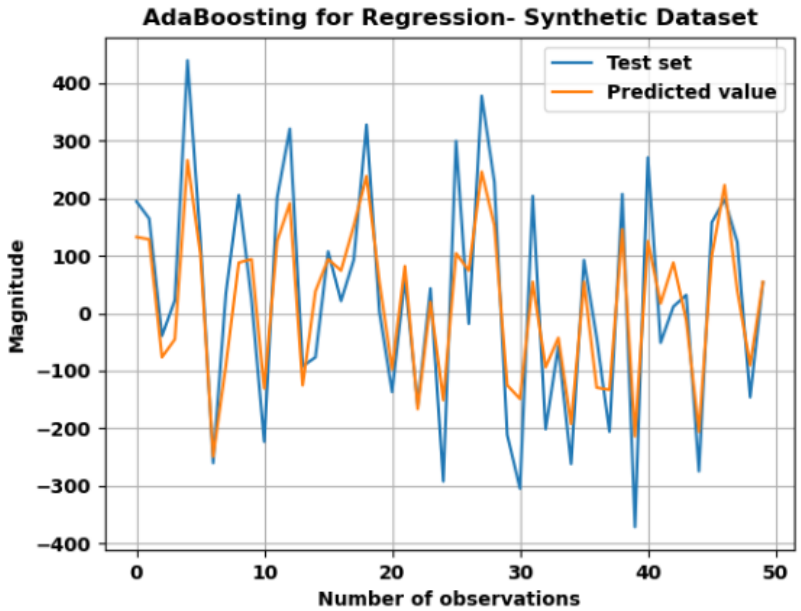
1. make_regression() function to create a synthetic binary classification dataset.
2. Split the dataset into train-test $70 - 30\%$.
3. Define the model by calling AdaBoostClassifier()
4. Make a prediction using the trained model.
5. Display classification metrics.

Adaptive Boosting for Regression

```python
import matplotlib.pyplot as plt
from sklearn.datasets import make_regression
from sklearn.ensemble import AdaBoostRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_log_error
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.metrics import median_absolute_error
from sklearn.metrics import max_error
from sklearn.metrics import explained_variance_score

X, y = make_regression(n_samples=1000, n_features=20, n_informative=15, noise=0.1, random_state=6)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
print(X.shape, y.shape)

model = AdaBoostRegressor()
model.fit(X_train,y_train)

y_hat = model.predict(X_test)

# evaluate the model and Metrics
print(f'The Mean Absolute Error is : {mean_absolute_error(y_test,y_hat):.3f}')
print(f'The Mean Squared error is : {mean_squared_error(y_test,y_hat):.3f}')
# print(f'The Mean Squared logarithmic error is : {mean_squared_log_error(y_test,y_hat):.3f}')
print(f'The Mean Absolute Percentage error is : {mean_absolute_percentage_error(y_test,y_hat):.3f}')
print(f'The Median Absolute % error is : {mean_absolute_percentage_error(y_test,y_hat):.3f}%')
print(f'The Median Absolute error is : {median_absolute_error(y_test,y_hat):.3f}')
print(f'The Max error is : {max_error(y_test,y_hat):.3f}')
print(f'The Explained variance score (R-squared) is : {100*explained_variance_score(y_test,y_hat):.3f}%')
```

Classification Metrics

- Mean Absolute Error (MAE) : 72.58
- Mean Squared Error (MSE) : 8626.864
- Mean Absolute Percentage Error: 5.085%
- Median Absolute error : 59.557
- Max Error : 271.175
- The Explained variance score (R^2) : 73.02%
- The Adjusted ($\overline{R}^2$) : 69.436%



Regression Metrics

Mean Absolute Error (MAE)

A risk metric corresponding to the expected value of the absolute error loss or L_1-norm loss.

$$MAE(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} |y_i - \hat{y}_i|$$

Mean Squared Error (MSE)

A risk metric corresponding to the expected value of the squared (quadratic) error loss or loss.

$$MSE(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} (y_i - \hat{y}_i)^2$$

Regression Metrics

Mean squared logarithmic error (MSLE)

A risk metric corresponding to the expected value of the squared logarithmic error or loss.

$$MSLE(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} (\log_e(1 + y_i) - \log_e(1 + \hat{y}_i))^2$$

Mean Absolute Percentage Error(MAPE)

is an evaluation metric for regression problems. The idea of this metric is to be sensitive to relative errors. It is for example not changed by a global scaling of the target variable.

$$MAPE(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} \frac{|y_i - \hat{y}_i|}{\max(\epsilon, |y_i|)}$$

Regression Metrics

Median Absolute Error (MedAE)

is particularly interesting because it is robust to outliers. The loss is calculated by taking the median of all absolute differences between the target and the prediction.

$$MedAE(y, \hat{y}) = median(|y_1 - \hat{y}_1|, ..., |y_n - \hat{y}_n|)$$

Max Error

a metric that captures the worst case error between the predicted value and the true value.

$$MaxError(y, \hat{y}) = \max(|y_i - \hat{y}_i|)$$

Regression Metrics

Explained variance score (R^2)

Or coefficient of determination.

$$Explained\_variance = 1 - \frac{Var\{y - \hat{y}\}}{Var\{y\}}$$

Adjusted $\overline{R}^2$

$$Adjusted\_R^2 = 1 - \frac{(1 - R^2)(n_{samples} - 1)}{n_{samples} - p}$$

where p is the # of features and $n_{samples}$ is the row of dataset used for train or test.

Stacked Generalization

- Stacked generalization or stacking is an ensemble machine learning algorithm to combine multiple classification via a meta-classifier.

Stacked Generalization

- Stacked generalization or stacking is an ensemble machine learning algorithm to combine multiple classification via a meta-classifier.
- The individual classification models are trained on the complete training set.

Stacked Generalization

- Stacked generalization or stacking is an ensemble machine learning algorithm to combine multiple classification via a meta-classifier.
- The individual classification models are trained on the complete training set.
- Stacking attack a learning problem with different type of models which are capable to learn some part of problem but not the whole space of problem.

Stacked Generalization

- Stacked generalization or stacking is an ensemble machine learning algorithm to combine multiple classification via a meta-classifier.
- The individual classification models are trained on the complete training set.
- Stacking attack a learning problem with different type of models which are capable to learn some part of problem but not the whole space of problem.
- Build multiple different model which learns from the intermediate predictions, one prediction for each learned model.

Stacked Generalization

- Stacked generalization or stacking is an ensemble machine learning algorithm to combine multiple classification via a meta-classifier.
- The individual classification models are trained on the complete training set.
- Stacking attack a learning problem with different type of models which are capable to learn some part of problem but not the whole space of problem.
- Build multiple different model which learns from the intermediate predictions, one prediction for each learned model.
- Then add a new model which learns from the intermediate predictions the same target.

Stacked Generalization

- Explore a space of different models for the same problem.

Stacked Generalization

- Explore a space of different models for the same problem.
- The final model is called Meta-classifier to be stacked on the top of the others.

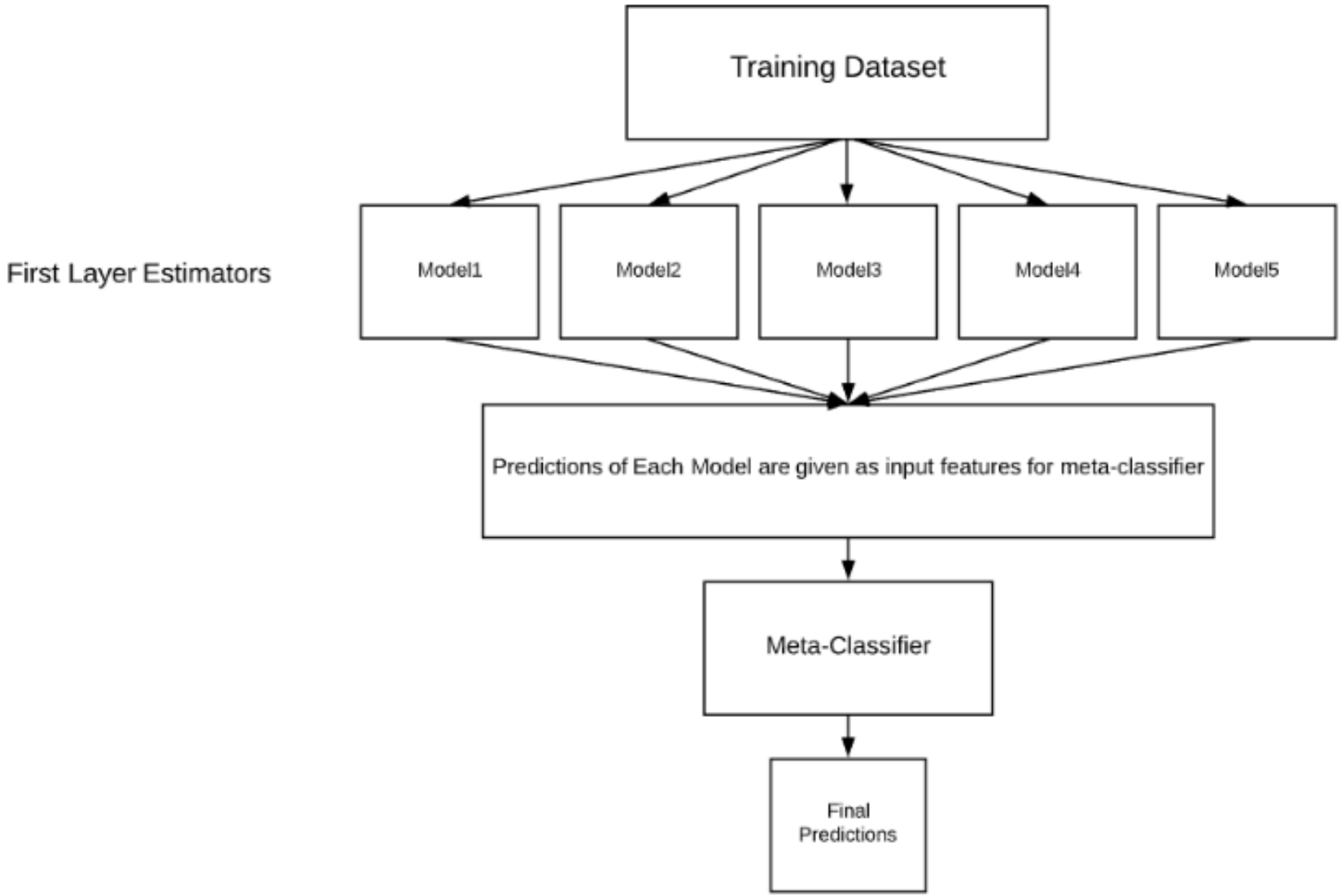
Stacked Generalization

- Explore a space of different models for the same problem.
- The final model is called Meta-classifier to be stacked on the top of the others.
- Stacking might improve the overall performance, and often end up with a model which is better than any individual intermediate model.

Stacked Generalization

- Explore a space of different models for the same problem.
- The final model is called Meta-classifier to be stacked on the top of the others.
- Stacking might improve the overall performance, and often end up with a model which is better than any individual intermediate model.
- Improved performance by stacking is NOT guaranteed as it often the case with any machine learning technique.

Stacking Architecture



Stacking and Python Implementing

1. Heart.cvs dataset with 14 features and 1025 instances.

Final Results

Stacking and Python Implementing

1. Heart.cvs dataset with 14 features and 1025 instances.
2. Loading dataset and preparation.

Final Results

Stacking and Python Implementing

1. Heart.cvs dataset with 14 features and 1025 instances.
2. Loading dataset and preparation.
3. Split the dataset into train-test $80 - 20\%$.

Final Results

Stacking and Python Implementing

1. Heart.cvs dataset with 14 features and 1025 instances.
2. Loading dataset and preparation.
3. Split the dataset into train-test $80 - 20\%$.
4. Standardizing data.

Final Results

Stacking and Python Implementing

1. Heart.cvs dataset with 14 features and 1025 instances.
2. Loading dataset and preparation.
3. Split the dataset into train-test $80-20\%$.
4. Standardizing data.
5. Building firs layer classifier (KNeighborsClassifier()), train and evaluate accuracy on the test set.

Final Results

Stacking and Python Implementing

1. Heart.cvs dataset with 14 features and 1025 instances.
2. Loading dataset and preparation.
3. Split the dataset into train-test $80 - 20\%$.
4. Standardizing data.
5. Building firs layer classifier (KNeighborsClassifier()), train and evaluate accuracy on the test set.
6. Building second layer classifier (Naive Bayes Classifier), train and evaluate accuracy on the test set.

Final Results

Stacking and Python Implementing

1. Heart.cvs dataset with 14 features and 1025 instances.
2. Loading dataset and preparation.
3. Split the dataset into train-test $80 - 20\%$.
4. Standardizing data.
5. Building firs layer classifier (KNeighborsClassifier()), train and evaluate accuracy on the test set.
6. Building second layer classifier (Naive Bayes Classifier), train and evaluate accuracy on the test set.
7. Implementing stacking classifier and evaluating the accuracy on the stacking classifier.

Final Results

Stacking and Python Implementing

1. Heart.cvs dataset with 14 features and 1025 instances.
2. Loading dataset and preparation.
3. Split the dataset into train-test $80 - 20\%$.
4. Standardizing data.
5. Building firs layer classifier (KNeighborsClassifier()), train and evaluate accuracy on the test set.
6. Building second layer classifier (Naive Bayes Classifier), train and evaluate accuracy on the test set.
7. Implementing stacking classifier and evaluating the accuracy on the stacking classifier.

Final Results

- Accuracy of Naive Bayes Classifier: 80.0%

Stacking and Python Implementing

1. Heart.cvs dataset with 14 features and 1025 instances.
2. Loading dataset and preparation.
3. Split the dataset into train-test $80 - 20\%$.
4. Standardizing data.
5. Building firs layer classifier (KNeighborsClassifier()), train and evaluate accuracy on the test set.
6. Building second layer classifier (Naive Bayes Classifier), train and evaluate accuracy on the test set.
7. Implementing stacking classifier and evaluating the accuracy on the stacking classifier.

Final Results

- Accuracy of Naive Bayes Classifier: 80.0%
- Accuracy of KNeighbors Classifier is: 80.48%

Stacking and Python Implementing

1. Heart.cvs dataset with 14 features and 1025 instances.
2. Loading dataset and preparation.
3. Split the dataset into train-test $80 - 20\%$.
4. Standardizing data.
5. Building firs layer classifier (KNeighborsClassifier()), train and evaluate accuracy on the test set.
6. Building second layer classifier (Naive Bayes Classifier), train and evaluate accuracy on the test set.
7. Implementing stacking classifier and evaluating the accuracy on the stacking classifier.

Final Results

- Accuracy of Naive Bayes Classifier: 80.0%
- Accuracy of KNeighbors Classifier is: 80.48%
- Accuracy of Stacked model: 83.90%

Stacking-Python-Sample Code

```
import pandas as pd
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_confusion_matrix
from mlxtend.classifier import StackingClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

df = pd.read_csv('heart.csv')    # loading the dataset
print(df.to_string())
X = df.drop('target', axis=1)
y = df['target']
# 20 % training dataset is considered for testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

Stacking-Python-Sample Code

```
NB = GaussianNB()                # initialising Naive Bayes
model_NaiveBayes = NB.fit(X_train, y_train)
pred_nb = model_NaiveBayes.predict(X_test)
acc_nb = accuracy_score(y_test, pred_nb)
print('Accuracy of Naive Bayes Classifier:', acc_nb * 100)

KNC = KNeighborsClassifier()   # initialising KNeighbors Classifier
model_kNeighborsClassifier = KNC.fit(X_train, y_train)   # fitting Training Set
pred_knc = model_kNeighborsClassifier.predict(X_test)   # Predicting on test dataset

acc_knc = accuracy_score(y_test, pred_knc)  # evaluating accuracy score
print('accuracy score of KNeighbors Classifier is:', acc_knc * 100)

lr = LogisticRegression() # defining meta-classifier
clf_stack = StackingClassifier(classifiers =[KNC, NB], meta_classifier = lr,
                               use_probas = True, use_features_in_secondary = True)

model_stack = clf_stack.fit(X_train, y_train)   # training of stacked model
pred_stack = model_stack.predict(X_test)       # predictions on test data using stacked model

acc_stack = accuracy_score(y_test, pred_stack)  # evaluating accuracy
print('accuracy score of Stacked model:', acc_stack * 100)
```