**Ensemble Learning and Random Forests**

* A group of predictors is called **ensemble,** group of multiple predictors or models are called **Ensemble Learning** & Ensemble Learning Algorithm is called as **Ensemble Method.**

# Voting Classifiers

* Aggregate the predictions of each classifier & predict the class that gets the most votes.
* The Class that gets the highest vote is the final output of the ensemble. This majority- vote classifier is called as **Hard Voting Classifier.**

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import make\_moons

X,y = make\_moons(n\_samples=500,noise=0.3,random\_state=42)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=42)

Plt.scatter(X[:0],X[:1],c=y)

from sklearn.ensamble import RandomForestClassifier

from sklearn.ensamble import VotingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

log\_clf = LogisticRegression(random\_state=42)

rnd\_clf = RandomForestClassifier(random\_state=42)

svc\_clf = SVC(random\_state=42)

voting\_clf = VotingClassifier(

estimators = [(‘lr’,log\_clf),(‘rf’,rnd\_clf),(‘svc’,svc\_clf)],

voting = ‘hard’)

voting\_clf.fit(X\_train,y\_train)

from sklearn.metrics import accuracy\_score

for clf in (log\_clf, rnd\_clf, svm\_clf, voting\_clf):

clf.fit(X\_train,y\_train)

y\_pred = clf.predict(X\_test)

print(clf.\_\_class\_\_.\_\_name\_\_, accuracy\_score(y\_test, y\_pred))

* **Soft Voting** : if all classifiers are able to estimate class probabilities (predict\_proba() method), then Scikit-Learn can predict (voting = ‘soft’) the class with the highest average probability. It often archives higher performance in comparison to Hard Voting.

log\_clf = LogisticRegression(random\_state=42)

rnd\_clf = RandomForestClassifier(random\_state=42)

svc\_clf = SVC(random\_state=42)

voting\_clf = VotingClassifier(

estimators =[(‘lr’,log\_clf),(‘rf’,rnd\_clf),(‘svc’,svc\_clf)],

voting = ‘soft’)

voting\_clf.fit(X\_train,y\_train)

**#Accuracy Code**

res = {}

**for** (name, prd) **in** ('lr', log\_clf), ('rf', rnd\_clf), ('svc', svm\_clf):

predictions = prd.predict\_proba(X\_test)

res[name] = predictions

correct = 0

wrong = 0

**for** i **in** range(len(y\_test)):

avgprob = (res['lr'][i] + res['rf'][i] + res['svc'][i])/3

outcome = 0

**if** avgprob[0] < avgprob[1]:

outcome = (res['lr'][i] + res['rf'][i] + res['svc'][i]) > 1

**if** outcome == y\_test[i]: correct += 1

**else**: wrong += 1

print(correct/(correct + wrong))

**#Accuracy Measure using Standard method**

**for** clf **in** (log\_clf, rnd\_clf, svm\_clf, voting\_clf):

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

print(clf.\_\_class\_\_.\_\_name\_\_, accuracy\_score(y\_test, y\_pred))

# Bagging and Pasting

* Ensemble methods performs best when a diverse set of classifiers are used. There are two ways to achieve this objective :
  + Use different Classifier Algorithm
  + Use the same Algorithm, but train them on different random subset of training data.
* **Bagging :** When sampling is performed with replacement.
* **Pasting :** When Sampling is performed without replacement.
* **Bagging or Pasting** can be trained in parallel.
* **Scikit Learn** offers a simple API **BaggingClassifier (or BaggingRegressor).** Set ‘**bootstrap = True’** for **Bagging. ‘n\_jobs’** parameter will tell how many CPU cores needs to be used. **= -1** means use all available cores.
* Then **BaggingClassifier Algo** automatically performs **Soft Voting** in case base classifier can estimate **Class Probabilities(predict\_proba())**
* Overall Bagging Classifier performs better results. However, Pasting is good for large data-sets.
* It has comparable bias but a smaller variance.

from sklearn.ensemble import BaggingClassifier

from sklearn.tree import DecisionTreeClassifier

bag\_clf = BaggingClassifier(

DecisionTreeClassifier(),

n\_estimators = 500, **#No of trees**

max\_samples =300,

bootstrap = False,

n\_jobs = -1

)

bag\_clf.fit(X\_train,y\_train)

y\_pred = bag\_clf.predict(X\_test)

accuracy\_score(y\_pred,y\_test)

# Out-of-Bag Evaluation

* With sampling, some instances may have picked multiple times while others are not picked at all. It is around 63%. The remaining 37% that are picked are called as **out-of-bag(oob) instances**.
* So, these instances can be used for evaluation. Setting **obb\_score = True** in **BaggingClassifier,**  evaluation score is available by **obb\_score\_** variable.

bag\_clf = BaggingClassifier (

DecisionTreeClassifier(),

n\_estimators = 500,

bootstrap = True,

random\_state=42,

n\_jobs = -1,

obb\_score = True)

bag\_clf.fit(X\_train,y\_train)

bag\_clf.obb\_score\_

bag\_clf.obb\_decision\_function\_

* BaggingClassifier also supports sampling the Features. It is controlled by 2 hyperparameters **max\_features & bootstrap\_features.**
* Sampling both instances and features is known as **Random Patches Method.** Sampling only features is known as **Random Subspaces method.**

# Random Forests

* Random Forests is an ensemble of Decision Trees, generally trained by bagging method with max\_samples = size of the training set.
* Instead of searching for the best nodes to splitting the nodes, it searches for best features among a random subset of features.

from sklearn.ensemble import RandomForestClassifier

rnd\_clf = RandomForestClassifier(n\_estimators = 500, max\_leaf\_nodes=16,

n\_jobs = -1)

rnd\_clf.fit(X\_train,y\_train)

y\_pred\_rnd = rnd\_clf.predict(X\_test)

rnd\_clf.obb\_score\_

rnd\_clf.feature\_importances\_

# Random Forests – Extra Trees

* When growing a tree in a Random Forest, at each node only a random subset of features is considered for splitting. It is possible to make trees more random also by using random thresholds for each feature rather than searching for the best possible thresholds. A forest of such extremely random trees is called as **Extremely Randomized Tree Ensemble or Extra-Trees.**
* It trades for more bias for a lower variance. It makes Extra-Trees much faster.