**Ensemble Learning**

**Learning Objectives**

By the end of this lesson, you will be able to:

Define ensemble learning

List different types of ensemble methods

Build an intuition

Apply different algorithms of ensemble learning using use cases

**What Is Ensemble Learning?**

Ensemble techniques combine individual models to improve the stability and predictive power of the model.

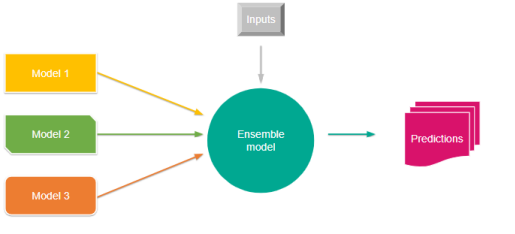
**Ideology Behind Ensemble Learning:**

Certain models do well in modeling one aspect of the data, while others do well in modeling another.

Instead of learning a single complex model, learn several simple models and combine their output to produce the final decision. Individual model variances and biases are balanced by the strength of other models in ensemble learning.

Ensemble learning will provide a composite prediction where the final accuracy is better than the accuracy of individual models.

**Working of Ensemble Learning**

****

**Significance of Ensemble Learning**

Robustness

Ensemble models incorporate the predictions from all the base learners

Accuracy

Ensemble models deliver accurate predictions and have improved performances

**Ensemble Learning Methods**

Techniques for creating an ensemble model

Combine all weak learners to form an ensemble, or create an ensemble of well-chosen strong and diverse models

**Steps Involved in Ensemble Methods**

Every ensemble algorithm consists of two steps:

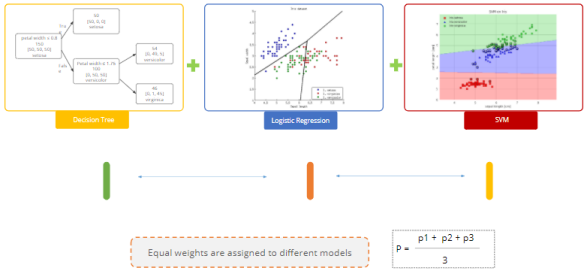
Producing a cohort of predictions using simple ML algorithms

Combining the predictions into one aggregated model

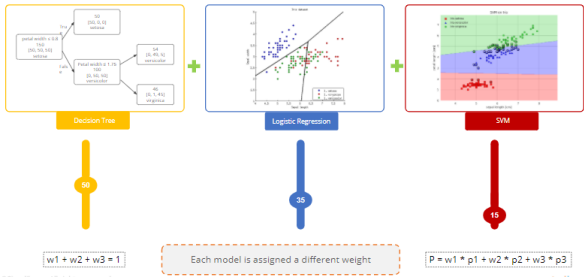
The ensemble can be achieved through several techniques.

**Types of Ensemble Methods**

**Averaging**

****

**Weighted Averaging**

**Bagging Algorithms**

Bootstrap Aggregation or bagging involves taking multiple samples from your training dataset (with replacement) and training a model for each sample.

The final output prediction is averaged across the predictions of all of the submodels.

The three bagging models covered in this section are as follows:

Bagged Decision Trees

Random Forest

Extra Trees

**1. Bagged Decision Trees**

Bagging performs best with algorithms that have a high variance. A popular example is decision trees, often constructed without pruning.

Below, you can see an example of using the BaggingClassifier with the Classification and Regression Trees algorithm (DecisionTreeClassifier). A total of 100 trees are created.

Scikit-learn is a Python library that provides a consistent interface for machine learning and statistical modeling, including classification, regression, clustering, and dimensionality reduction.

Pandas is a Python library for data manipulation and analysis.

In [71]:*#Bagged Decision Trees for Classification*

**import** pandas

**from** sklearn **import** model\_selection

**from** sklearn.ensemble **import** BaggingClassifier

**from** sklearn.tree **import** DecisionTreeClassifier

url **=** "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv" names **=** ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

dataframe **=** pandas**.**read\_csv(url, names**=**names)

array **=** dataframe**.**values

X **=** array[:,0:8]

Y **=** array[:,8]

seed **=** 7

kfold **=** model\_selection**.**KFold(n\_splits**=**10, random\_state**=**seed, shuffle**=True**)

cart **=** DecisionTreeClassifier()

num\_trees **=** 100

model **=** BaggingClassifier(base\_estimator**=**cart, n\_estimators**=**num\_trees, random\_state**=**seed)

results **=** model\_selection**.**cross\_val\_score(model, X, Y, cv**=**kfold)

print(results**.**mean())

0.7578263841421736

**2. Random Forest**

Random forest is an extension of bagged decision trees.

Samples of the training dataset are taken with replacement, but the trees are constructed in a way that reduces the correlation between individual classifiers. Specifically, rather than greedily choosing the best split point in the construction of the tree, only a random subset of features is considered for each split.

You can construct a Random Forest model for classification using the RandomForestClassifier class.

The example below provides a sample of Random Forest for classification with 100 trees and split points chosen from a random selection of three features.

In [72]:*#Random Forest Classification*

**import** pandas

**from** sklearn **import** model\_selection

**from** sklearn.ensemble **import** RandomForestClassifier

url **=** "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv" names **=** ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

dataframe **=** pandas**.**read\_csv(url, names**=**names)

array **=** dataframe**.**values

X **=** array[:,0:8]

Y **=** array[:,8]

seed **=** 7

num\_trees **=** 100

max\_features **=** 3

kfold **=** model\_selection**.**KFold(n\_splits**=**10, random\_state**=**seed, shuffle**=True**)

model **=** RandomForestClassifier(n\_estimators**=**num\_trees, max\_features**=**max\_features)

results **=** model\_selection**.**cross\_val\_score(model, X, Y, cv**=**kfold)

print(results**.**mean())

0.764354066985646

**3. Extra Trees**

Extra Trees are another modification of bagging where random trees are constructed from samples of the training dataset. You can construct an Extra Trees model for classification using the ExtraTreesClassifier class.

The example below provides a demonstration of extra trees with a tree set of 100 and splits chosen from seven random features.

In [73]:*#Extra Trees Classification*

**import** pandas

**from** sklearn **import** model\_selection

**from** sklearn.ensemble **import** ExtraTreesClassifier

url **=** "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv" names **=** ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

dataframe **=** pandas**.**read\_csv(url, names**=**names)

array **=** dataframe**.**values

X **=** array[:,0:8]

Y **=** array[:,8]

seed **=** 7

num\_trees **=** 100

max\_features **=** 7

kfold **=** model\_selection**.**KFold(n\_splits**=**10, random\_state**=**seed, shuffle**=True**)

model **=** ExtraTreesClassifier(n\_estimators**=**num\_trees, max\_features**=**max\_features)

results **=** model\_selection**.**cross\_val\_score(model, X, Y, cv**=**kfold)

print(results**.**mean())

0.7577922077922079

**Boosting Algorithms**

Boosting ensemble algorithms create a sequence of models that attempts to correct the mistakes of the models before them in the sequence.

Once created, the models make predictions that may be weighted by their demonstrated accuracy, and the results are combined to create a final output prediction.

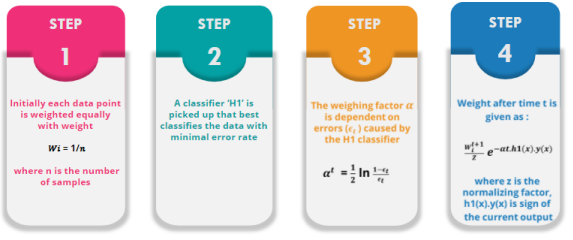
The two most common boosting ensemble machine learning algorithms are:

AdaBoost

Stochastic Gradient Boosting

**AdaBoost**

AdaBoost was the first successful boosting ensemble algorithm. It generally works by weighting instances in the dataset by how easy or difficult they are to classify, allowing the algorithm to pay more or less attention to them in the construction of subsequent models.

You can construct an AdaBoost model for classification using the AdaBoostClassifier class.

The example below demonstrates the construction of 30 decision trees in sequence using the AdaBoost algorithm.

In [74]:*#AdaBoost Classification*

**import** pandas

**from** sklearn **import** model\_selection

**from** sklearn.ensemble **import** AdaBoostClassifier

url **=** "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv" names **=** ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

dataframe **=** pandas**.**read\_csv(url, names**=**names)

array **=** dataframe**.**values

X **=** array[:,0:8]

Y **=** array[:,8]

seed **=** 7

num\_trees **=** 30

kfold **=** model\_selection**.**KFold(n\_splits**=**10, random\_state**=**seed, shuffle**=True**)

model **=** AdaBoostClassifier(n\_estimators**=**num\_trees, random\_state**=**seed)

results **=** model\_selection**.**cross\_val\_score(model, X, Y, cv**=**kfold)

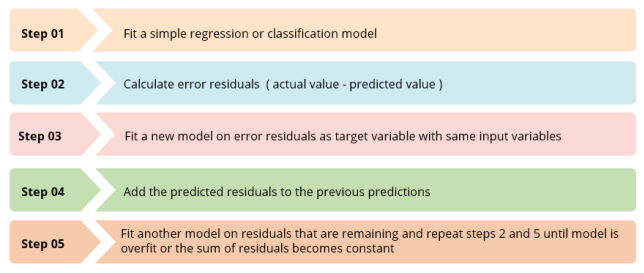
print(results**.**mean())

0.7552802460697198

**Stochastic Gradient Boosting**

One of the most advanced ensemble approaches is Stochastic Gradient Boosting (also known as Gradient Boosting Machines). It's also a strategy that's proven to be one of the most effective methods for boosting performance via ensemble.

**Steps of Gradient Boasting Machine**

****You can construct a Gradient Boosting model for classification using the **GradientBoostingClassifier** class.

The example below demonstrates Stochastic Gradient Boosting for classification with 100 trees.

In [75]:*#Stochastic Gradient Boosting Classification*

**import** pandas

**from** sklearn **import** model\_selection

**from** sklearn.ensemble **import** GradientBoostingClassifier

url **=** "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv" names **=** ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

dataframe **=** pandas**.**read\_csv(url, names**=**names)

array **=** dataframe**.**values

X **=** array[:,0:8]

Y **=** array[:,8]

seed **=** 7

num\_trees **=** 100

kfold **=** model\_selection**.**KFold(n\_splits**=**10, random\_state**=**seed, shuffle**=True**)

model **=** GradientBoostingClassifier(n\_estimators**=**num\_trees, random\_state**=**seed)

results **=** model\_selection**.**cross\_val\_score(model, X, Y, cv**=**kfold)

print(results**.**mean())

0.7591934381408066

**CatBoost**

CatBoost is an algorithm for gradient boosting on decision trees. It is developed by Yandex researchers and engineers and is used for search, recommendation systems, personal assistants, self-driving cars, weather prediction, and many other tasks at Yandex and in other companies, including CERN, Cloudflare, Careem taxi. It is open-source and can be used by anyone.

Let's study this with the help of a use case.

**Data Description**

The data consists of real historical data collected from 2010 & 2011. Employees are manually allowed or denied access to resources over time. You must create an algorithm capable of learning from this historical data to predict approval or denial for an unknown set of employees.

**File Descriptions**

**train.csv:** It is a training set. Each row has the action (ground truth), resources, and information about the employee's role at the time of approval.

**test.csv:** It is the test set for which predictions should be made. Each row asks whether an employee having the listed characteristics should have access to the listed resource.

The objective is to develop a model from historical data that will decide the access needs of an employee so that manual access transactions (grants and revocations) are reduced as the attributes of the employee change over time. The model will take information on the position of an employee and a resource code and return whether access should be given or not.

Note: The problem statement is from a Kaggle contest

The objective is to develop a model from historical data, that will decide the access needs of an employee, so that manual access transactions (grants and revocations) are reduced as the attributes of the employee change over time. The model will take information on the position of an employee and a resource code and return whether access should be given or not.

Note: The problem statement is from a Kaggle contest

**Libraries Installation**

In [76]:*#Installing CatBoost*

**!**pip install catboost

Requirement already satisfied: catboost in c:\users\alpika.gupta\anaconda3\lib\site-packages (1.0.6) Requirement already satisfied: graphviz in c:\users\alpika.gupta\anaconda3\lib\site-packages (from catboost) (0.20.1)

Requirement already satisfied: pandas>=0.24.0 in c:\users\alpika.gupta\anaconda3\lib\site-packages (from catboo st) (1.3.4)

Requirement already satisfied: numpy>=1.16.0 in c:\users\alpika.gupta\anaconda3\lib\site-packages (from catboos t) (1.20.3)

Requirement already satisfied: scipy in c:\users\alpika.gupta\anaconda3\lib\site-packages (from catboost) (1.7. 1)

Requirement already satisfied: six in c:\users\alpika.gupta\anaconda3\lib\site-packages (from catboost) (1.16. 0)

Requirement already satisfied: matplotlib in c:\users\alpika.gupta\anaconda3\lib\site-packages (from catboost) (3.4.3)

Requirement already satisfied: plotly in c:\users\alpika.gupta\anaconda3\lib\site-packages (from catboost) (5. 8.0)

Requirement already satisfied: pytz>=2017.3 in c:\users\alpika.gupta\anaconda3\lib\site-packages (from pandas>= 0.24.0->catboost) (2021.3)

Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\alpika.gupta\anaconda3\lib\site-packages (fro m pandas>=0.24.0->catboost) (2.8.2)

Requirement already satisfied: cycler>=0.10 in c:\users\alpika.gupta\anaconda3\lib\site-packages (from matplotl ib->catboost) (0.10.0)

Requirement already satisfied: pyparsing>=2.2.1 in c:\users\alpika.gupta\anaconda3\lib\site-packages (from matp lotlib->catboost) (3.0.4)

Requirement already satisfied: pillow>=6.2.0 in c:\users\alpika.gupta\anaconda3\lib\site-packages (from matplot lib->catboost) (8.4.0)

Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\alpika.gupta\anaconda3\lib\site-packages (from mat plotlib->catboost) (1.3.1)

Requirement already satisfied: tenacity>=6.2.0 in c:\users\alpika.gupta\anaconda3\lib\site-packages (from plotl y->catboost) (8.0.1)

In [77]:*#To import libraries*

**import** catboost

print(catboost**.**\_\_version\_\_)

**!**python --version

1.0.6

Python 3.9.7

**Reading the Data**

In [78]:*#To read the data*

**import** pandas **as** pd

**import** os

**import** numpy **as** np

np**.**set\_printoptions(precision**=**4)

**import** catboost

**from** catboost **import \***

**from** catboost **import** datasets

In [79]:(train\_df, test\_df) **=** catboost**.**datasets**.**amazon()

In [80]:train\_df**.**head()

Out[80]:

**ACTION RESOURCE MGR\_ID ROLE\_ROLLUP\_1 ROLE\_ROLLUP\_2 ROLE\_DEPTNAME ROLE\_TITLE ROLE\_FAMILY\_DESC ROLE\_FAMILY ROLE\_CO 0** 1 39353 85475 117961 118300 123472 117905 117906 290919 1179 **1** 1 17183 1540 117961 118343 123125 118536 118536 308574 1185 **2** 1 36724 14457 118219 118220 117884 117879 267952 19721 1178 **3** 1 36135 5396 117961 118343 119993 118321 240983 290919 1183 **4** 1 42680 5905 117929 117930 119569 119323 123932 19793 1193

The data will be displayed on the screen.

**Preparing Your Data**

Label values extraction

Action column contains the categorical feature. However, it is not available for test dataset, so you must drop the Action column.

In [81]:y **=** train\_df**.**ACTION

X **=** train\_df**.**drop('ACTION', axis**=**1)

Categorical features declaration

**cat\_features** is a one-dimensional array of categorical columns indices.

It has one of the following types: list, numpy.ndarray, pandas.DataFrame, and pandas.Series.

Now we will declare the cat feature that holds the categorical values present on train dataset.

In [82]:*#The type list is used here*

cat\_features **=** list(range(0, X**.**shape[1]))

print(cat\_features)

[0, 1, 2, 3, 4, 5, 6, 7, 8]

In [83]:*#looking for label balance in dataset*

print('Labels: {}'**.**format(set(y)))

print('Zero count = {}, One count = {}'**.**format(len(y) **-** sum(y), sum(y)))

Labels: {0, 1}

Zero count = 1897, One count = 30872

Ways to create **Pool** class

In multiprocessing, the Pool class may handle a huge number of processes. It enables you to run several jobs in a single process due to its ability to queue the jobs.

In [84]:*#Specifying the dataset*

dataset\_dir **=** './amazon'

**if not** os**.**path**.**exists(dataset\_dir):

os**.**makedirs(dataset\_dir)

*#We will be able to work with files with/without header and with different separators*

train\_df**.**to\_csv(

os**.**path**.**join(dataset\_dir, 'train.tsv'),

index**=False**, sep**=**'\t', header**=False**

)

test\_df**.**to\_csv(

os**.**path**.**join(dataset\_dir, 'test.tsv'),

index**=False**, sep**=**'\t', header**=False**

)

train\_df**.**to\_csv(

os**.**path**.**join(dataset\_dir, 'train.csv'),

index**=False**, sep**=**',', header**=True**

)

test\_df**.**to\_csv(

os**.**path**.**join(dataset\_dir, 'test.csv'),

index**=False**, sep**=**',', header**=True**

)

In [85]:**!**head amazon/train.csv

'head' is not recognized as an internal or external command,

operable program or batch file.

In [86]:**from** catboost.utils **import** create\_cd

feature\_names **=** dict()

**for** column, name **in** enumerate(train\_df):

**if** column **==** 0:

**continue**

feature\_names[column **-** 1] **=** name

create\_cd(

label**=**0,

cat\_features**=**list(range(1, train\_df**.**columns**.**shape[0])),

feature\_names**=**feature\_names,

output\_path**=**os**.**path**.**join(dataset\_dir, 'train.cd')

)

In [87]:**!**cat amazon/train.cd

'cat' is not recognized as an internal or external command,

operable program or batch file.

In [88]:pool1 **=** Pool(data**=**X, label**=**y, cat\_features**=**cat\_features)

pool2 **=** Pool(

data**=**os**.**path**.**join(dataset\_dir, 'train.csv'),

delimiter**=**',',

column\_description**=**os**.**path**.**join(dataset\_dir, 'train.cd'),

has\_header**=True**

)

pool3 **=** Pool(data**=**X, cat\_features**=**cat\_features)

*#Fastest way to create a Pool is to create it from numpy matrix.*

*#This way should be used if you want fast predictions*

*#or fastest way to load the data in python.*

X\_prepared **=** X**.**values**.**astype(str)**.**astype(object)

*#For FeaturesData class categorial features must have type str*

pool4 **=** Pool(

data**=**FeaturesData(

cat\_feature\_data**=**X\_prepared,

cat\_feature\_names**=**list(X)

),

label**=**y**.**values

)

print('Dataset shape')

print('dataset 1:' **+** str(pool1**.**shape) **+**

'\ndataset 2:' **+** str(pool2**.**shape) **+**

'\ndataset 3:' **+** str(pool3**.**shape) **+**

'\ndataset 4: ' **+** str(pool4**.**shape))

print('\n')

print('Column names')

print('dataset 1:')

print(pool1**.**get\_feature\_names())

print('\ndataset 2:')

print(pool2**.**get\_feature\_names())

print('\ndataset 3:')

print(pool3**.**get\_feature\_names())

print('\ndataset 4:')

print(pool4**.**get\_feature\_names())

Dataset shape

dataset 1:(32769, 9)

dataset 2:(32769, 9)

dataset 3:(32769, 9)

dataset 4: (32769, 9)

Column names

dataset 1:

['RESOURCE', 'MGR\_ID', 'ROLE\_ROLLUP\_1', 'ROLE\_ROLLUP\_2', 'ROLE\_DEPTNAME', 'ROLE\_TITLE', 'ROLE\_FAMILY\_DESC', 'RO LE\_FAMILY', 'ROLE\_CODE']

dataset 2:

['RESOURCE', 'MGR\_ID', 'ROLE\_ROLLUP\_1', 'ROLE\_ROLLUP\_2', 'ROLE\_DEPTNAME', 'ROLE\_TITLE', 'ROLE\_FAMILY\_DESC', 'RO LE\_FAMILY', 'ROLE\_CODE']

dataset 3:

['RESOURCE', 'MGR\_ID', 'ROLE\_ROLLUP\_1', 'ROLE\_ROLLUP\_2', 'ROLE\_DEPTNAME', 'ROLE\_TITLE', 'ROLE\_FAMILY\_DESC', 'RO LE\_FAMILY', 'ROLE\_CODE']

dataset 4:

['RESOURCE', 'MGR\_ID', 'ROLE\_ROLLUP\_1', 'ROLE\_ROLLUP\_2', 'ROLE\_DEPTNAME', 'ROLE\_TITLE', 'ROLE\_FAMILY\_DESC', 'RO LE\_FAMILY', 'ROLE\_CODE']

**Split Your Data into Train and Validation**

Let us split the data into **Train** and **Validation**.

In [89]:**from** sklearn.model\_selection **import** train\_test\_split

X\_train, X\_validation, y\_train, y\_validation **=** train\_test\_split(X, y, train\_size**=**0.8, random\_state**=**1234)

**Selecting the Objective Function**

Possible options for binary classification:

Logloss

CrossEntropy for probabilities in target

A **CatBoostClassifier** trains and applies models for the classification problems. It provides compatibility with the scikit-learn tools.

In [20]:**from** catboost **import** CatBoostClassifier

model **=** CatBoostClassifier(

iterations**=**5,

learning\_rate**=**0.1,

*#loss\_function='CrossEntropy'*

)

model**.**fit(

X\_train, y\_train,

cat\_features**=**cat\_features,

eval\_set**=**(X\_validation, y\_validation),

verbose**=False**

)

print('Model is fitted: ' **+** str(model**.**is\_fitted()))

print('Model params:')

print(model**.**get\_params())

Model is fitted: True

Model params:

{'iterations': 5, 'learning\_rate': 0.1}

**Stdout of the Training**

Stdout displays output directly to the screen console. Output can take any form. It can be output from a print statement, an expression statement, or even a direct prompt.

In [21]:**from** catboost **import** CatBoostClassifier

model **=** CatBoostClassifier(

iterations**=**15,

*#verbose=5,*

)

model**.**fit(

X\_train, y\_train,

cat\_features**=**cat\_features,

eval\_set**=**(X\_validation, y\_validation),

)

Learning rate set to 0.441257

0: learn: 0.4220777 test: 0.4223741 best: 0.4223741 (0) total: 33.2ms remaining: 465ms 1: learn: 0.3149660 test: 0.3151186 best: 0.3151186 (1) total: 80.7ms remaining: 524ms 2: learn: 0.2621494 test: 0.2629766 best: 0.2629766 (2) total: 113ms remaining: 453ms 3: learn: 0.2302316 test: 0.2302315 best: 0.2302315 (3) total: 149ms remaining: 410ms 4: learn: 0.2060274 test: 0.2019603 best: 0.2019603 (4) total: 180ms remaining: 361ms 5: learn: 0.1956107 test: 0.1894627 best: 0.1894627 (5) total: 217ms remaining: 326ms 6: learn: 0.1870345 test: 0.1790904 best: 0.1790904 (6) total: 251ms remaining: 287ms 7: learn: 0.1836943 test: 0.1748030 best: 0.1748030 (7) total: 281ms remaining: 246ms 8: learn: 0.1807119 test: 0.1707896 best: 0.1707896 (8) total: 314ms remaining: 210ms 9: learn: 0.1775777 test: 0.1662489 best: 0.1662489 (9) total: 347ms remaining: 174ms 10: learn: 0.1762130 test: 0.1654446 best: 0.1654446 (10) total: 378ms remaining: 137ms 11: learn: 0.1760650 test: 0.1653191 best: 0.1653191 (11) total: 391ms remaining: 97.8ms

12: learn: 0.1748232 test: 0.1642093 best: 0.1642093 (12) total: 425ms remaining: 65.3ms 13: learn: 0.1742028 test: 0.1638902 best: 0.1638902 (13) total: 456ms remaining: 32.5ms 14: learn: 0.1733966 test: 0.1627237 best: 0.1627237 (14) total: 485ms remaining: 0us

bestTest = 0.162723674

bestIteration = 14

Out[21]:

<catboost.core.CatBoostClassifier at 0x269b437ea00>

**Metric Calculation and Graph Plotting**

Let us perform metric calculation and graph plotting by importing the **CatBoostClassifier**.

In [22]:**from** catboost **import** CatBoostClassifier model **=** CatBoostClassifier(

iterations**=**50,

random\_seed**=**63,

learning\_rate**=**0.5,

custom\_loss**=**['AUC', 'Accuracy']

)

model**.**fit(

X\_train, y\_train,

cat\_features**=**cat\_features,

eval\_set**=**(X\_validation, y\_validation), verbose**=False**,

plot**=True**

)

Out[22]:

<catboost.core.CatBoostClassifier at 0x269b437efd0>

**Model Comparison**

Let us compare the models.

In [23]:model1 **=** CatBoostClassifier(

learning\_rate**=**0.7,

iterations**=**100,

random\_seed**=**0,

train\_dir**=**'learing\_rate\_0.7'

)

model2 **=** CatBoostClassifier(

learning\_rate**=**0.01,

iterations**=**100,

random\_seed**=**0,

train\_dir**=**'learing\_rate\_0.01'

)

model1**.**fit(

X\_train, y\_train,

eval\_set**=**(X\_validation, y\_validation), cat\_features**=**cat\_features,

verbose**=False**

)

model2**.**fit(

X\_train, y\_train,

eval\_set**=**(X\_validation, y\_validation), cat\_features**=**cat\_features,

verbose**=False**

)

Out[23]:

<catboost.core.CatBoostClassifier at 0x269b790d550>

In [24]:**from** catboost **import** MetricVisualizer

MetricVisualizer(['learing\_rate\_0.01', 'learing\_rate\_0.7'])**.**start()

**Best Iteration**

In [25]:*#Performing best iteration*

**from** catboost **import** CatBoostClassifier

model **=** CatBoostClassifier(

iterations**=**100,

random\_seed**=**63,

learning\_rate**=**0.5,

*#use\_best\_model=False*

)

model**.**fit(

X\_train, y\_train,

cat\_features**=**cat\_features,

eval\_set**=**(X\_validation, y\_validation),

verbose**=False**,

plot**=True**

)

Out[25]:

<catboost.core.CatBoostClassifier at 0x269b45e6a60>

In [26]:print('Tree count: ' **+** str(model**.**tree\_count\_))

Tree count: 82

**Cross-Validation**

Cross-validation is a technique which involves reserving a particular sample of a dataset on which you do not train the model. CatBoost allows to perform cross-validation on the given dataset.

In [27]:*#Performing cross-validation*

**from** catboost **import** cv

params **=** {}

params['loss\_function'] **=** 'Logloss'

params['iterations'] **=** 80

params['custom\_loss'] **=** 'AUC'

params['random\_seed'] **=** 63

params['learning\_rate'] **=** 0.5

cv\_data **=** cv(

params **=** params,

pool **=** Pool(X, label**=**y, cat\_features**=**cat\_features),

fold\_count**=**5,

shuffle**=True**,

partition\_random\_seed**=**0,

plot**=True**,

stratified**=False**,

verbose**=False**

)

Training on fold [0/5]

bestTest = 0.1695893693

bestIteration = 38

Training on fold [1/5]

bestTest = 0.164632916

bestIteration = 48

Training on fold [2/5]

bestTest = 0.1515742763

bestIteration = 60

Training on fold [3/5]

bestTest = 0.1426916182

bestIteration = 78

Training on fold [4/5]

bestTest = 0.1563234371

bestIteration = 37

In [28]:cv\_data**.**head()

Out[28]:

**iterations test-Logloss-mean test-Logloss-std train-Logloss-mean train-Logloss-std test-AUC-mean test-AUC-std 0** 0 0.302367 0.004317 0.302196 0.004517 0.513577 0.030360 **1** 1 0.227370 0.007679 0.228497 0.005126 0.642263 0.048004 **2** 2 0.190856 0.006917 0.196796 0.003999 0.791709 0.011361 **3** 3 0.178884 0.007455 0.186682 0.003242 0.813889 0.009362 **4** 4 0.172286 0.007957 0.181380 0.002135 0.826529 0.005319

**Logloss** is indicative of how close the prediction probability is to the corresponding true value. Let us print the **Best validation Logloss score**.

In [29]:best\_value **=** np**.**min(cv\_data['test-Logloss-mean'])

best\_iter **=** np**.**argmin(cv\_data['test-Logloss-mean'])

print('Best validation Logloss score, not stratified: {:.4f}±{:.4f} on step {}'**.**format(

best\_value,

cv\_data['test-Logloss-std'][best\_iter],

best\_iter)

)

Best validation Logloss score, not stratified: 0.1581±0.0104 on step 52

In [30]:cv\_data **=** cv(

params **=** params,

pool **=** Pool(X, label**=**y, cat\_features**=**cat\_features),

fold\_count**=**5,

type **=** 'Classical',

shuffle**=True**,

partition\_random\_seed**=**0,

plot**=True**,

stratified**=True**,

verbose**=False**

)

best\_value **=** np**.**min(cv\_data['test-Logloss-mean'])

best\_iter **=** np**.**argmin(cv\_data['test-Logloss-mean'])

print('Best validation Logloss score, stratified: {:.4f}±{:.4f} on step {}'**.**format(

best\_value,

cv\_data['test-Logloss-std'][best\_iter],

best\_iter)

)

Training on fold [0/5]

bestTest = 0.1614486451

bestIteration = 31

Training on fold [1/5]

bestTest = 0.1554856763

bestIteration = 57

Training on fold [2/5]

bestTest = 0.1588065247

bestIteration = 46

Training on fold [3/5]

bestTest = 0.1525713791

bestIteration = 60

Training on fold [4/5]

bestTest = 0.1576264978

bestIteration = 29

Best validation Logloss score, stratified: 0.1579±0.0036 on step 57

**Overfitting Detector**

If overfitting occurs, CatBoost can stop the training earlier than the training parameters dictate. For example, it can be stopped before the specified number of trees are built. This option is set in the starting parameters.

In [31]:model\_with\_early\_stop **=** CatBoostClassifier(

iterations**=**200,

random\_seed**=**63,

learning\_rate**=**0.5,

early\_stopping\_rounds**=**20

)

model\_with\_early\_stop**.**fit(

X\_train, y\_train,

cat\_features**=**cat\_features,

eval\_set**=**(X\_validation, y\_validation),

verbose**=False**,

plot**=True**

)

Out[31]:

<catboost.core.CatBoostClassifier at 0x269b7926460>

In [32]:print(model\_with\_early\_stop**.**tree\_count\_) 30

In [33]:model\_with\_early\_stop **=** CatBoostClassifier( eval\_metric**=**'AUC',

iterations**=**200,

random\_seed**=**63,

learning\_rate**=**0.5,

early\_stopping\_rounds**=**20

)

model\_with\_early\_stop**.**fit(

X\_train, y\_train,

cat\_features**=**cat\_features,

eval\_set**=**(X\_validation, y\_validation), verbose**=False**,

plot**=True**

)

Out[33]:

<catboost.core.CatBoostClassifier at 0x269b7926f40>

In [34]:print(model\_with\_early\_stop**.**tree\_count\_)

30

**Select Decision Boundary**

In classification problems with two or more classes, a decision boundary is a hypersurface that separates the underlying vector space into sets, keeping one for each class.

In [35]:model **=** CatBoostClassifier(

random\_seed**=**63,

iterations**=**200,

learning\_rate**=**0.03,

)

model**.**fit(

X\_train, y\_train,

cat\_features**=**cat\_features,

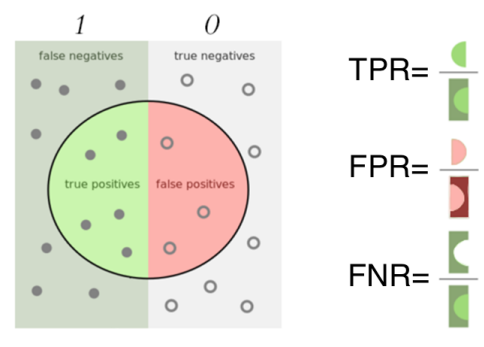
verbose**=False**,

plot**=True**

)

Out[35]:

<catboost.core.CatBoostClassifier at 0x269b7926b80>



In [36]:*#Using utils to make the pattern easier*

**from** catboost.utils **import** get\_roc\_curve

**import** sklearn

**from** sklearn **import** metrics

eval\_pool **=** Pool(X\_validation, y\_validation, cat\_features**=**cat\_features)

curve **=** get\_roc\_curve(model, eval\_pool)

(fpr, tpr, thresholds) **=** curve

roc\_auc **=** sklearn**.**metrics**.**auc(fpr, tpr)

In [37]:**import** matplotlib.pyplot **as** plt

plt**.**figure(figsize**=**(16, 8))

lw **=** 2

plt**.**plot(fpr, tpr, color**=**'darkorange',

lw**=**lw, label**=**'ROC curve (area = %0.2f)' **%** roc\_auc, alpha**=**0.5)

plt**.**plot([0, 1], [0, 1], color**=**'navy', lw**=**lw, linestyle**=**'--', alpha**=**0.5)

plt**.**xlim([0.0, 1.0])

plt**.**ylim([0.0, 1.05])

plt**.**xticks(fontsize**=**16)

plt**.**yticks(fontsize**=**16)

plt**.**grid(**True**)

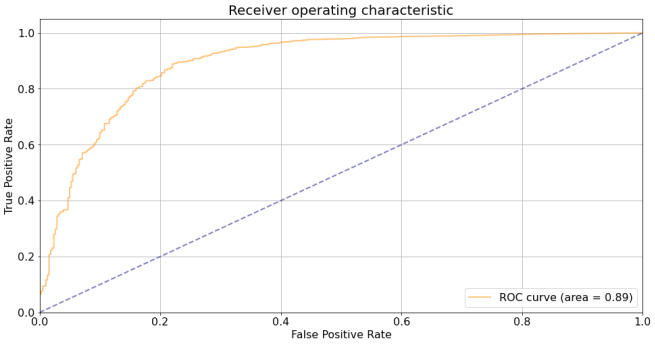
plt**.**xlabel('False Positive Rate', fontsize**=**16)

plt**.**ylabel('True Positive Rate', fontsize**=**16)

plt**.**title('Receiver operating characteristic', fontsize**=**20)

plt**.**legend(loc**=**"lower right", fontsize**=**16)

plt**.**show()

The above graph illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.

In [38]:**from** catboost.utils **import** get\_fpr\_curve

**from** catboost.utils **import** get\_fnr\_curve

(thresholds, fpr) **=** get\_fpr\_curve(curve**=**curve)

(thresholds, fnr) **=** get\_fnr\_curve(curve**=**curve)

In [39]:plt**.**figure(figsize**=**(16, 8))

lw **=** 2

plt**.**plot(thresholds, fpr, color**=**'blue', lw**=**lw, label**=**'FPR', alpha**=**0.5)

plt**.**plot(thresholds, fnr, color**=**'green', lw**=**lw, label**=**'FNR', alpha**=**0.5)

plt**.**xlim([0.0, 1.0])

plt**.**ylim([0.0, 1.05])

plt**.**xticks(fontsize**=**16)

plt**.**yticks(fontsize**=**16)

plt**.**grid(**True**)

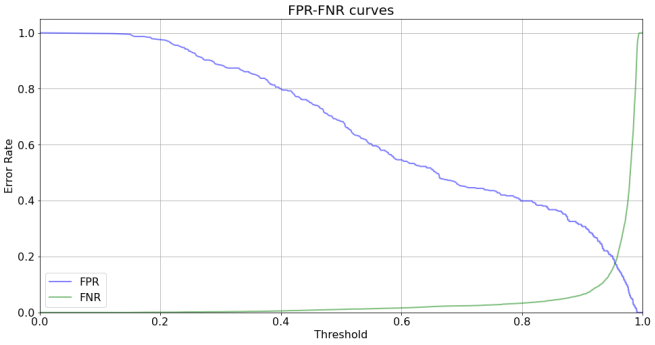
plt**.**xlabel('Threshold', fontsize**=**16)

plt**.**ylabel('Error Rate', fontsize**=**16)

plt**.**title('FPR-FNR curves', fontsize**=**20)

plt**.**legend(loc**=**"lower left", fontsize**=**16)

plt**.**show()

The above graph displays the FPR-FNR curves for error rate and threshold.

In [40]:**from** catboost.utils **import** select\_threshold

print(select\_threshold(model**=**model, data**=**eval\_pool, FNR**=**0.01))

print(select\_threshold(model**=**model, data**=**eval\_pool, FPR**=**0.01))

0.48034745289964775

0.9899643475743053

**Snapshotting**

Catboost supports snapshotting. You can use it to recover training after an interruption or start training with previous results.

In [41]:*#!rm 'catboost\_info/snapshot.bkp'*

**from** catboost **import** CatBoostClassifier

model **=** CatBoostClassifier(

iterations**=**100,

save\_snapshot**=True**,

snapshot\_file**=**'snapshot.bkp',

snapshot\_interval**=**1,

random\_seed**=**43

)

model**.**fit(

X\_train, y\_train,

eval\_set**=**(X\_validation, y\_validation),

cat\_features**=**cat\_features,

verbose**=True**

)

Learning rate set to 0.193326

0: learn: 0.5565905 test: 0.5566217 best: 0.5566217 (0) total: 17.3ms remaining: 1.71s 1: learn: 0.4642626 test: 0.4639935 best: 0.4639935 (1) total: 51ms remaining: 2.5s 2: learn: 0.3989148 test: 0.3981304 best: 0.3981304 (2) total: 79.1ms remaining: 2.56s 3: learn: 0.3516186 test: 0.3510286 best: 0.3510286 (3) total: 92.4ms remaining: 2.22s 4: learn: 0.3164302 test: 0.3161297 best: 0.3161297 (4) total: 124ms remaining: 2.35s 5: learn: 0.2906047 test: 0.2905494 best: 0.2905494 (5) total: 134ms remaining: 2.09s 6: learn: 0.2710475 test: 0.2708899 best: 0.2708899 (6) total: 157ms remaining: 2.09s 7: learn: 0.2538458 test: 0.2539798 best: 0.2539798 (7) total: 194ms remaining: 2.23s

8: learn: 0.2399269 test: 0.2401350 best: 0.2401350 (8) total: 237ms remaining: 2.4s 9: learn: 0.2298664 test: 0.2304173 best: 0.2304173 (9) total: 271ms remaining: 2.44s 10: learn: 0.2180381 test: 0.2161946 best: 0.2161946 (10) total: 303ms remaining: 2.45s 11: learn: 0.2089276 test: 0.2055572 best: 0.2055572 (11) total: 336ms remaining: 2.46s 12: learn: 0.2027605 test: 0.1985313 best: 0.1985313 (12) total: 361ms remaining: 2.42s 13: learn: 0.1980597 test: 0.1929079 best: 0.1929079 (13) total: 395ms remaining: 2.42s 14: learn: 0.1931802 test: 0.1869335 best: 0.1869335 (14) total: 430ms remaining: 2.44s 15: learn: 0.1899584 test: 0.1827767 best: 0.1827767 (15) total: 475ms remaining: 2.49s 16: learn: 0.1868109 test: 0.1791481 best: 0.1791481 (16) total: 510ms remaining: 2.49s 17: learn: 0.1844046 test: 0.1764212 best: 0.1764212 (17) total: 544ms remaining: 2.48s 18: learn: 0.1826815 test: 0.1739992 best: 0.1739992 (18) total: 576ms remaining: 2.45s 19: learn: 0.1813182 test: 0.1724242 best: 0.1724242 (19) total: 608ms remaining: 2.43s 20: learn: 0.1799044 test: 0.1705979 best: 0.1705979 (20) total: 648ms remaining: 2.44s 21: learn: 0.1787567 test: 0.1691593 best: 0.1691593 (21) total: 699ms remaining: 2.48s 22: learn: 0.1778479 test: 0.1679138 best: 0.1679138 (22) total: 732ms remaining: 2.45s 23: learn: 0.1769800 test: 0.1669261 best: 0.1669261 (23) total: 772ms remaining: 2.44s 24: learn: 0.1761963 test: 0.1659238 best: 0.1659238 (24) total: 822ms remaining: 2.47s 25: learn: 0.1759394 test: 0.1656044 best: 0.1656044 (25) total: 842ms remaining: 2.4s 26: learn: 0.1756321 test: 0.1651183 best: 0.1651183 (26) total: 881ms remaining: 2.38s 27: learn: 0.1753093 test: 0.1645482 best: 0.1645482 (27) total: 913ms remaining: 2.35s 28: learn: 0.1746490 test: 0.1634569 best: 0.1634569 (28) total: 949ms remaining: 2.32s

29: learn: 0.1741248 test: 0.1626719 best: 0.1626719 (29) total: 984ms remaining: 2.3s 30: learn: 0.1739190 test: 0.1624505 best: 0.1624505 (30) total: 1.02s remaining: 2.27s 31: learn: 0.1737505 test: 0.1621577 best: 0.1621577 (31) total: 1.08s remaining: 2.29s 32: learn: 0.1735156 test: 0.1617652 best: 0.1617652 (32) total: 1.12s remaining: 2.27s 33: learn: 0.1733737 test: 0.1616706 best: 0.1616706 (33) total: 1.17s remaining: 2.27s 34: learn: 0.1730710 test: 0.1615700 best: 0.1615700 (34) total: 1.21s remaining: 2.24s 35: learn: 0.1726234 test: 0.1613597 best: 0.1613597 (35) total: 1.24s remaining: 2.21s 36: learn: 0.1724223 test: 0.1609771 best: 0.1609771 (36) total: 1.28s remaining: 2.18s 37: learn: 0.1723166 test: 0.1609138 best: 0.1609138 (37) total: 1.31s remaining: 2.14s 38: learn: 0.1721109 test: 0.1607760 best: 0.1607760 (38) total: 1.35s remaining: 2.11s 39: learn: 0.1718253 test: 0.1604844 best: 0.1604844 (39) total: 1.38s remaining: 2.07s 40: learn: 0.1718135 test: 0.1604838 best: 0.1604838 (40) total: 1.41s remaining: 2.03s 41: learn: 0.1717654 test: 0.1604337 best: 0.1604337 (41) total: 1.46s remaining: 2.02s

42: learn: 0.1714937 test: 0.1602021 best: 0.1602021 (42) total: 1.51s remaining: 2s 43: learn: 0.1713759 test: 0.1601088 best: 0.1601088 (43) total: 1.54s remaining: 1.96s 44: learn: 0.1712930 test: 0.1601280 best: 0.1601088 (43) total: 1.57s remaining: 1.92s 45: learn: 0.1710363 test: 0.1601333 best: 0.1601088 (43) total: 1.61s remaining: 1.89s 46: learn: 0.1708310 test: 0.1598021 best: 0.1598021 (46) total: 1.66s remaining: 1.87s 47: learn: 0.1707415 test: 0.1598379 best: 0.1598021 (46) total: 1.69s remaining: 1.84s 48: learn: 0.1706194 test: 0.1597803 best: 0.1597803 (48) total: 1.73s remaining: 1.8s 49: learn: 0.1705094 test: 0.1598181 best: 0.1597803 (48) total: 1.77s remaining: 1.77s 50: learn: 0.1703454 test: 0.1596611 best: 0.1596611 (50) total: 1.81s remaining: 1.74s 51: learn: 0.1701364 test: 0.1594569 best: 0.1594569 (51) total: 1.86s remaining: 1.71s 52: learn: 0.1699351 test: 0.1594792 best: 0.1594569 (51) total: 1.89s remaining: 1.68s 53: learn: 0.1698191 test: 0.1594453 best: 0.1594453 (53) total: 1.93s remaining: 1.65s 54: learn: 0.1696360 test: 0.1594233 best: 0.1594233 (54) total: 1.97s remaining: 1.61s 55: learn: 0.1695476 test: 0.1593201 best: 0.1593201 (55) total: 2.02s remaining: 1.59s 56: learn: 0.1694507 test: 0.1593702 best: 0.1593201 (55) total: 2.05s remaining: 1.55s 57: learn: 0.1693879 test: 0.1593558 best: 0.1593201 (55) total: 2.1s remaining: 1.52s 58: learn: 0.1692182 test: 0.1593365 best: 0.1593201 (55) total: 2.14s remaining: 1.49s 59: learn: 0.1691692 test: 0.1593046 best: 0.1593046 (59) total: 2.17s remaining: 1.45s 60: learn: 0.1689825 test: 0.1590959 best: 0.1590959 (60) total: 2.2s remaining: 1.41s 61: learn: 0.1688233 test: 0.1587949 best: 0.1587949 (61) total: 2.24s remaining: 1.37s 62: learn: 0.1687423 test: 0.1587121 best: 0.1587121 (62) total: 2.28s remaining: 1.34s 63: learn: 0.1685821 test: 0.1585367 best: 0.1585367 (63) total: 2.31s remaining: 1.3s 64: learn: 0.1682095 test: 0.1583598 best: 0.1583598 (64) total: 2.35s remaining: 1.27s 65: learn: 0.1680435 test: 0.1581916 best: 0.1581916 (65) total: 2.39s remaining: 1.23s 66: learn: 0.1678092 test: 0.1580685 best: 0.1580685 (66) total: 2.42s remaining: 1.19s 67: learn: 0.1676905 test: 0.1579741 best: 0.1579741 (67) total: 2.45s remaining: 1.16s 68: learn: 0.1674648 test: 0.1579143 best: 0.1579143 (68) total: 2.49s remaining: 1.12s 69: learn: 0.1673214 test: 0.1579064 best: 0.1579064 (69) total: 2.53s remaining: 1.08s 70: learn: 0.1672664 test: 0.1579522 best: 0.1579064 (69) total: 2.57s remaining: 1.05s 71: learn: 0.1672555 test: 0.1579174 best: 0.1579064 (69) total: 2.63s remaining: 1.02s 72: learn: 0.1672462 test: 0.1579299 best: 0.1579064 (69) total: 2.69s remaining: 996ms 73: learn: 0.1670767 test: 0.1578301 best: 0.1578301 (73) total: 2.73s remaining: 958ms 74: learn: 0.1670546 test: 0.1577969 best: 0.1577969 (74) total: 2.76s remaining: 920ms 75: learn: 0.1669987 test: 0.1577948 best: 0.1577948 (75) total: 2.8s remaining: 885ms 76: learn: 0.1668860 test: 0.1577237 best: 0.1577237 (76) total: 2.84s remaining: 847ms 77: learn: 0.1667215 test: 0.1576140 best: 0.1576140 (77) total: 2.9s remaining: 818ms 78: learn: 0.1667140 test: 0.1575968 best: 0.1575968 (78) total: 2.94s remaining: 781ms 79: learn: 0.1666534 test: 0.1575803 best: 0.1575803 (79) total: 2.99s remaining: 748ms 80: learn: 0.1666215 test: 0.1575678 best: 0.1575678 (80) total: 3.03s remaining: 710ms 81: learn: 0.1665369 test: 0.1576578 best: 0.1575678 (80) total: 3.06s remaining: 673ms 82: learn: 0.1664881 test: 0.1576933 best: 0.1575678 (80) total: 3.11s remaining: 638ms 83: learn: 0.1664809 test: 0.1576931 best: 0.1575678 (80) total: 3.16s remaining: 602ms 84: learn: 0.1664459 test: 0.1577250 best: 0.1575678 (80) total: 3.21s remaining: 566ms 85: learn: 0.1664193 test: 0.1577061 best: 0.1575678 (80) total: 3.24s remaining: 528ms 86: learn: 0.1663895 test: 0.1577540 best: 0.1575678 (80) total: 3.29s remaining: 491ms 87: learn: 0.1663837 test: 0.1577521 best: 0.1575678 (80) total: 3.32s remaining: 453ms 88: learn: 0.1663628 test: 0.1577793 best: 0.1575678 (80) total: 3.36s remaining: 416ms 89: learn: 0.1663101 test: 0.1577507 best: 0.1575678 (80) total: 3.4s remaining: 378ms 90: learn: 0.1662302 test: 0.1576135 best: 0.1575678 (80) total: 3.44s remaining: 340ms 91: learn: 0.1662257 test: 0.1576185 best: 0.1575678 (80) total: 3.47s remaining: 302ms 92: learn: 0.1662070 test: 0.1576770 best: 0.1575678 (80) total: 3.52s remaining: 265ms 93: learn: 0.1661519 test: 0.1576925 best: 0.1575678 (80) total: 3.57s remaining: 228ms 94: learn: 0.1659885 test: 0.1576700 best: 0.1575678 (80) total: 3.62s remaining: 190ms 95: learn: 0.1659752 test: 0.1577138 best: 0.1575678 (80) total: 3.66s remaining: 153ms 96: learn: 0.1658729 test: 0.1577104 best: 0.1575678 (80) total: 3.7s remaining: 114ms 97: learn: 0.1658312 test: 0.1577005 best: 0.1575678 (80) total: 3.73s remaining: 76.2ms 98: learn: 0.1657567 test: 0.1576310 best: 0.1575678 (80) total: 3.77s remaining: 38.1ms 99: learn: 0.1655047 test: 0.1576241 best: 0.1575678 (80) total: 3.81s remaining: 0us

bestTest = 0.1575677776

bestIteration = 80

Shrink model to first 81 iterations.

Out[41]:

<catboost.core.CatBoostClassifier at 0x269c0f10700>

**Model Predictions**

predict\_proba gives you the probabilities for the target in array form. The number of probabilities for each row is equal to the number of categories in the target variable.

In [42]:print(model**.**predict\_proba(X**=**X\_validation))

[[0.0508 0.9492]

[0.0181 0.9819]

[0.0179 0.9821]

...

[0.0161 0.9839]

[0.017 0.983 ]

[0.0236 0.9764]]

In [43]:print(model**.**predict(data**=**X\_validation))

[1 1 1 ... 1 1 1]

In [44]:raw\_pred **=** model**.**predict(

data**=**X\_validation,

prediction\_type**=**'RawFormulaVal'

)

print(raw\_pred)

[2.9282 3.9947 4.0077 ... 4.1115 4.06 3.7207]

In [45]:**from** numpy **import** exp

*#Calculating sigmoid*

sigmoid **= lambda** x: 1 **/** (1 **+** exp(**-**x))

probabilities **=** sigmoid(raw\_pred)

print(probabilities)

[0.9492 0.9819 0.9821 ... 0.9839 0.983 0.9764]

The probabilities will be displayed on the screen.

In [46]:X\_prepared **=** X\_validation**.**values**.**astype(str)**.**astype(object)

*#For FeaturesData class categorial features must have type str*

fast\_predictions **=** model**.**predict\_proba(

X**=**FeaturesData(

cat\_feature\_data**=**X\_prepared,

cat\_feature\_names**=**list(X\_validation)

)

)

print(fast\_predictions)

[[0.0508 0.9492]

[0.0181 0.9819]

[0.0179 0.9821]

...

[0.0161 0.9839]

[0.017 0.983 ]

[0.0236 0.9764]]

**Staged Prediction**

CatBoost allows to apply a trained model and calculate the results for each i-th tree of the model, taking into consideration only the trees in the range [0; i).

In [47]:predictions\_gen **=** model**.**staged\_predict\_proba(

data**=**X\_validation,

ntree\_start**=**0,

ntree\_end**=**5,

eval\_period**=**1

)

**try**:

**for** iteration, predictions **in** enumerate(predictions\_gen):

print('Iteration ' **+** str(iteration) **+** ', predictions:')

print(predictions)

**except** Exception:

**pass**

Iteration 0, predictions:

[[0.4154 0.5846]

[0.4154 0.5846]

[0.4154 0.5846]

...

[0.4154 0.5846]

[0.4154 0.5846]

[0.4154 0.5846]]

Iteration 1, predictions:

[[0.3476 0.6524]

[0.3476 0.6524]

[0.3476 0.6524]

...

[0.3476 0.6524]

[0.3476 0.6524]

[0.3476 0.6524]]

Iteration 2, predictions:

[[0.292 0.708 ]

[0.292 0.708 ]

[0.2978 0.7022]

...

[0.2978 0.7022]

[0.292 0.708 ]

[0.2978 0.7022]]

Iteration 3, predictions:

[[0.2485 0.7515]

[0.2485 0.7515]

[0.2538 0.7462]

...

[0.2538 0.7462]

[0.2485 0.7515]

[0.2538 0.7462]]

Iteration 4, predictions:

[[0.2126 0.7874]

[0.2126 0.7874]

[0.2173 0.7827]

...

[0.2173 0.7827]

[0.2126 0.7874]

[0.2173 0.7827]]

**Solving Multiclass Classification Problem**

Let us solve the **Multiclass Classification Problem** using the **CatBoostClassifier.**

In [48]:**from** catboost **import** CatBoostClassifier

model **=** CatBoostClassifier(

iterations**=**50,

random\_seed**=**43,

loss\_function**=**'MultiClass'

)

model**.**fit(

X\_train, y\_train,

cat\_features**=**cat\_features,

eval\_set**=**(X\_validation, y\_validation),

verbose**=False**,

plot**=True**

)

Out[48]:

<catboost.core.CatBoostClassifier at 0x269b7ad3dc0>

For multiclass problems with many classes, sometimes, it's better to solve classification problems using ranking. To do that, we will build a dataset with groups. Every group will represent one object from our initial dataset. But it will have one additional categorical feature, a possible class value. Target values will be equal to 1 if the class value is equal to the correct class and 0 otherwise. Thus, each group will have exactly one 1 in labels and some zeros. You can put all possible class values in the group, or you can try setting only hard negatives if there are too many labels. We'll show this approach as an example of a binary classification problem.

In [49]:*#Defining custom function to build multiclass ranking*

**from** copy **import** deepcopy

**def** build\_multiclass\_ranking\_dataset(X, y, cat\_features, label\_values**=**[0,1], start\_group\_id**=**0): ranking\_matrix **=** []

ranking\_labels **=** []

group\_ids **=** []

X\_train\_matrix **=** X**.**values

y\_train\_vector **=** y**.**values

**for** obj\_idx **in** range(X**.**shape[0]):

obj **=** list(X\_train\_matrix[obj\_idx])

**for** label **in** label\_values:

obj\_of\_given\_class **=** deepcopy(obj)

obj\_of\_given\_class**.**append(label)

ranking\_matrix**.**append(obj\_of\_given\_class)

ranking\_labels**.**append(float(y\_train\_vector[obj\_idx] **==** label))

group\_ids**.**append(start\_group\_id **+** obj\_idx)

final\_cat\_features **=** deepcopy(cat\_features)

final\_cat\_features**.**append(X**.**shape[1]) *# new feature that we are adding should be categorical.* **return** Pool(ranking\_matrix, ranking\_labels, cat\_features**=**final\_cat\_features, group\_id **=** group\_ids)

In [50]:**from** catboost **import** CatBoost

params **=** {'iterations':150, 'learning\_rate':0.01, 'l2\_leaf\_reg':30, 'random\_seed':0, 'loss\_function':'QuerySoftMax'}

groupwise\_train\_pool **=** build\_multiclass\_ranking\_dataset(X\_train, y\_train, cat\_features, [0,1]) groupwise\_eval\_pool **=** build\_multiclass\_ranking\_dataset(X\_validation, y\_validation, cat\_features, [0,1], X\_train**.**shape[0])

model **=** CatBoost(params)

model**.**fit(

X**=**groupwise\_train\_pool,

verbose**=False**,

eval\_set**=**groupwise\_eval\_pool,

plot**=True**

)

Out[50]:

<catboost.core.CatBoost at 0x269af65cd90> Making predictions with ranking mode

In [51]:**import** math

obj **=** list(X\_validation**.**values[0])

ratings **=** []

**for** label **in** [0,1]:

obj\_with\_label **=** deepcopy(obj)

obj\_with\_label**.**append(label)

rating **=** model**.**predict([obj\_with\_label])[0]

ratings**.**append(rating)

print('Raw values:', np**.**array(ratings))

**def** soft\_max(values):

**return** [math**.**exp(val) **/** sum([math**.**exp(val) **for** val **in** values]) **for** val **in** values] print('Probabilities', np**.**array(soft\_max(ratings)))

Raw values: [-0.471 0.4713]

Probabilities [0.2804 0.7196]

**Metric Evaluation on a New Dataset**

Let us perform **Metric Evaluation** on a new dataset using the training data.

In [52]:model **=** CatBoostClassifier(

random\_seed**=**63,

iterations**=**200,

learning\_rate**=**0.03,

)

model**.**fit(

X\_train, y\_train,

cat\_features**=**cat\_features,

verbose**=**50

)

0: learn: 0.6569860 total: 31.6ms remaining: 6.28s

50: learn: 0.1907871 total: 2.33s remaining: 6.82s

100: learn: 0.1645125 total: 5.13s remaining: 5.03s

150: learn: 0.1565519 total: 8.4s remaining: 2.73s

199: learn: 0.1533854 total: 11.7s remaining: 0us

Out[52]:

<catboost.core.CatBoostClassifier at 0x269b4608a30>

In [53]:metrics **=** model**.**eval\_metrics(

data**=**pool1,

metrics**=**['Logloss','AUC'],

ntree\_start**=**0,

ntree\_end**=**0,

eval\_period**=**1,

plot**=True**

)

In [54]:print('AUC values:')

print(np**.**array(metrics['AUC']))

AUC values:

[0.4998 0.538 0.5504 0.5888 0.6536 0.6515 0.6476 0.648 0.7117 0.731

0.7277 0.7278 0.7299 0.7298 0.7275 0.7273 0.7336 0.735 0.7445 0.7606

0.7627 0.7627 0.7715 0.7699 0.7773 0.7824 0.7859 0.8067 0.818 0.846

0.8607 0.8651 0.874 0.8745 0.8797 0.8794 0.8964 0.8969 0.9042 0.9129

0.9154 0.916 0.9175 0.9197 0.9245 0.9253 0.9301 0.9298 0.9305 0.931

0.9316 0.9332 0.9333 0.9356 0.9361 0.938 0.9393 0.9392 0.9395 0.941

0.9417 0.9431 0.9433 0.9436 0.944 0.9452 0.9458 0.9458 0.9479 0.9492

0.9503 0.9509 0.9517 0.9527 0.9537 0.9541 0.955 0.9556 0.9559 0.9564

0.9574 0.958 0.9591 0.9598 0.9602 0.9606 0.961 0.9615 0.9621 0.9625

0.9629 0.9635 0.9641 0.9644 0.9646 0.965 0.9654 0.9657 0.9659 0.966

0.9662 0.9666 0.9668 0.9669 0.9673 0.9675 0.9677 0.9678 0.9679 0.9679

0.9681 0.9682 0.9682 0.9683 0.9684 0.9685 0.9686 0.9687 0.9687 0.9688

0.9688 0.9689 0.9689 0.9691 0.9692 0.9693 0.9693 0.9694 0.9694 0.9693

0.9694 0.9699 0.9704 0.9708 0.9712 0.9716 0.972 0.9721 0.9724 0.9724

0.9728 0.9731 0.9733 0.9736 0.9738 0.9738 0.9739 0.974 0.974 0.9742

0.9741 0.9744 0.9746 0.975 0.9751 0.9754 0.9756 0.9755 0.9759 0.9759

0.9762 0.9765 0.9765 0.9766 0.9767 0.9767 0.9768 0.9771 0.9771 0.9771

0.9772 0.9773 0.9775 0.9777 0.9777 0.9778 0.9779 0.9779 0.9779 0.9779

0.9779 0.9779 0.9779 0.9779 0.978 0.978 0.978 0.978 0.978 0.978

0.978 0.9783 0.9785 0.9785 0.9785 0.9785 0.9785 0.9785 0.9787 0.9788]

**Feature Importances**

**Feature importance** refers to techniques that assign a score to input features based on how useful they are at predicting a target variable.

In [55]:*#To find feature importance*

model**.**get\_feature\_importance(prettified**=True**)

Out[55]:

**Feature Id Importances**

**0** RESOURCE 22.311819

**1** MGR\_ID 18.121045

**2** ROLE\_DEPTNAME 14.891573

**3** ROLE\_ROLLUP\_2 12.310360

**4** ROLE\_CODE 10.388989

**5** ROLE\_FAMILY\_DESC 9.144182

**6** ROLE\_TITLE 6.402628

**7** ROLE\_FAMILY 4.649123

**8** ROLE\_ROLLUP\_1 1.780282

Scores are assigned to the input features.

**Feature Evaluation**

Let us perform feature evaluation using the **eval\_features( )** function.

In [56]:**from** catboost.eval.catboost\_evaluation **import \***

learn\_params **=** {'iterations': 20, *# 2000*

'learning\_rate': 0.5, *#we set big learning\_rate, because we have small iterations* 'random\_seed': 0,

'verbose': **False**,

'loss\_function' : 'Logloss',

'boosting\_type': 'Plain'}

evaluator **=** CatboostEvaluation('amazon/train.tsv',

fold\_size**=**10000, *#<= 50% of dataset*

fold\_count**=**20,

column\_description**=**'amazon/train.cd',

partition\_random\_seed**=**0,

*#working\_dir=...*

)

result **=** evaluator**.**eval\_features(learn\_config**=**learn\_params,

eval\_metrics**=**['Logloss', 'Accuracy'],

features\_to\_eval**=**[6, 7, 8])

In [57]:**from** catboost.eval.evaluation\_result **import \***

logloss\_result **=** result**.**get\_metric\_results('Logloss')

logloss\_result**.**get\_baseline\_comparison(

ScoreConfig(ScoreType**.**Rel, overfit\_iterations\_info**=False**)

)

Out[57]:

**PValue Score Quantile 0.005 Quantile 0.995 Decision Features: 6** 0.000189 1.010962 0.586856 1.396997 GOOD **Features: 7** 0.681322 -0.033237 -0.316979 0.280699 UNKNOWN **Features: 8** 0.005111 -0.439271 -0.812376 -0.114295 BAD

**Saving the Model**

In [58]:my\_best\_model **=** CatBoostClassifier(iterations**=**10)

my\_best\_model**.**fit(

X\_train, y\_train,

eval\_set**=**(X\_validation, y\_validation),

cat\_features**=**cat\_features,

verbose**=False**

)

my\_best\_model**.**save\_model('catboost\_model.bin')

my\_best\_model**.**save\_model('catboost\_model.json', format**=**'json')

In [59]:my\_best\_model**.**load\_model('catboost\_model.bin')

print(my\_best\_model**.**get\_params())

print(my\_best\_model**.**random\_seed\_)

{'iterations': 10, 'loss\_function': 'Logloss', 'verbose': 0}

0

**Hyperparameter Tunning**

Hyperparameter tuning is the process of determining the right combination of hyperparameters that allows the model to maximize model performance. Setting the correct combination of hyperparameters is the only way to extract the maximum performance out of models.

**Training Speed**

In [60]:**from** catboost **import** CatBoost

fast\_model **=** CatBoostClassifier(

random\_seed**=**63,

iterations**=**150,

learning\_rate**=**0.01,

boosting\_type**=**'Plain',

bootstrap\_type**=**'Bernoulli',

subsample**=**0.5,

one\_hot\_max\_size**=**20,

rsm**=**0.5,

leaf\_estimation\_iterations**=**5,

max\_ctr\_complexity**=**1)

fast\_model**.**fit(

X\_train, y\_train,

cat\_features**=**cat\_features,

verbose**=False**,

plot**=True**

)

Out[60]:

<catboost.core.CatBoostClassifier at 0x269b5f76400> **Accuracy**

In [61]:tunned\_model **=** CatBoostClassifier( random\_seed**=**63,

iterations**=**1000,

learning\_rate**=**0.03,

l2\_leaf\_reg**=**3,

bagging\_temperature**=**1,

random\_strength**=**1,

one\_hot\_max\_size**=**2,

leaf\_estimation\_method**=**'Newton'

)

tunned\_model**.**fit(

X\_train, y\_train,

cat\_features**=**cat\_features,

verbose**=False**,

eval\_set**=**(X\_validation, y\_validation), plot**=True**

)

Out[61]:

<catboost.core.CatBoostClassifier at 0x269b4b12430> **Training the Model after Parameter Tuning**

In [62]:best\_model **=** CatBoostClassifier(

random\_seed**=**63,

iterations**=**int(tunned\_model**.**tree\_count\_ **\*** 1.2),

)

best\_model**.**fit(

X, y,

cat\_features**=**cat\_features,

verbose**=**100

)

Learning rate set to 0.040343

0: learn: 0.6456466 total: 28.3ms remaining: 32.4s 100: learn: 0.1534434 total: 6.59s remaining: 1m 8s 200: learn: 0.1466718 total: 14.5s remaining: 1m 8s 300: learn: 0.1430361 total: 22.6s remaining: 1m 3s 400: learn: 0.1399976 total: 31.1s remaining: 57.8s 500: learn: 0.1372704 total: 40.6s remaining: 52.3s 600: learn: 0.1341051 total: 49.8s remaining: 45.1s 700: learn: 0.1313257 total: 59.1s remaining: 37.5s 800: learn: 0.1285775 total: 1m 7s remaining: 29.2s 900: learn: 0.1263064 total: 1m 16s remaining: 20.7s 1000: learn: 0.1238298 total: 1m 25s remaining: 12.3s 1100: learn: 0.1212612 total: 1m 33s remaining: 3.84s 1145: learn: 0.1202083 total: 1m 37s remaining: 0us

Out[62]:

<catboost.core.CatBoostClassifier at 0x269be0f3fd0> **Calculate Prediction**

In [63]:*#Let us calculate contest predictions*

X\_test **=** test\_df**.**drop('id', axis**=**1)

test\_pool **=** Pool(data**=**X\_test, cat\_features**=**cat\_features) contest\_predictions **=** best\_model**.**predict\_proba(test\_pool) print('Predictions:')

print(contest\_predictions)

Predictions:

[[0.4455 0.5545]

[0.0133 0.9867]

[0.0116 0.9884]

...

[0.0054 0.9946]

[0.0422 0.9578]

[0.0117 0.9883]]

**Voting Ensemble**

Voting is one of the simplest ways of combining the predictions from multiple machine learning algorithms.

It works by first creating two or more standalone models from your training dataset. A Voting Classifier can then be used to wrap your models and average the predictions of the submodels when asked to make predictions for new data.

The predictions of the submodels can be weighted, but specifying the weights for classifiers manually or even heuristically is difficult. More advanced methods can learn how to best weight the predictions from submodels, but this is called stacking (stacked generalization) and is currently not provided in scikit-learn.

You can create a voting ensemble model for classification using the **VotingClassifier** class.

The code below provides an example of combining the predictions of logistic regression, classification, and regression trees and support vector machines together for a classification problem.

In [64]:*#Voting Ensemble for Classification*

**import** pandas

**from** sklearn **import** model\_selection

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.tree **import** DecisionTreeClassifier

**from** sklearn.svm **import** SVC

**from** sklearn.ensemble **import** VotingClassifier

url **=** "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv" names **=** ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

dataframe **=** pandas**.**read\_csv(url, names**=**names)

array **=** dataframe**.**values

X **=** array[:,0:8]

Y **=** array[:,8]

seed **=** 7

kfold **=** model\_selection**.**KFold(n\_splits**=**10)

*#Create the sub models*

estimators **=** []

model1 **=** LogisticRegression()

estimators**.**append(('logistic', model1))

model2 **=** DecisionTreeClassifier()

estimators**.**append(('cart', model2))

model3 **=** SVC()

estimators**.**append(('svm', model3))

*#Create the ensemble model*

ensemble **=** VotingClassifier(estimators)

results **=** model\_selection**.**cross\_val\_score(ensemble, X, Y, cv**=**kfold)

print(results**.**mean())

C:\Users\alpika.gupta\Anaconda3\lib\site-packages\sklearn\linear\_model\\_logistic.py:444: ConvergenceWarning: lb fgs failed to converge (status=1):

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:

https://scikit-learn.org/stable/modules/preprocessing.html

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression

n\_iter\_i = \_check\_optimize\_result(

C:\Users\alpika.gupta\Anaconda3\lib\site-packages\sklearn\linear\_model\\_logistic.py:444: ConvergenceWarning: lb fgs failed to converge (status=1):

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:

https://scikit-learn.org/stable/modules/preprocessing.html

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression

n\_iter\_i = \_check\_optimize\_result(

C:\Users\alpika.gupta\Anaconda3\lib\site-packages\sklearn\linear\_model\\_logistic.py:444: ConvergenceWarning: lb fgs failed to converge (status=1):

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:

https://scikit-learn.org/stable/modules/preprocessing.html

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression

n\_iter\_i = \_check\_optimize\_result(

C:\Users\alpika.gupta\Anaconda3\lib\site-packages\sklearn\linear\_model\\_logistic.py:444: ConvergenceWarning: lb fgs failed to converge (status=1):

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:

https://scikit-learn.org/stable/modules/preprocessing.html

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression

n\_iter\_i = \_check\_optimize\_result(

C:\Users\alpika.gupta\Anaconda3\lib\site-packages\sklearn\linear\_model\\_logistic.py:444: ConvergenceWarning: lb fgs failed to converge (status=1):

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:

https://scikit-learn.org/stable/modules/preprocessing.html

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression

n\_iter\_i = \_check\_optimize\_result(

C:\Users\alpika.gupta\Anaconda3\lib\site-packages\sklearn\linear\_model\\_logistic.py:444: ConvergenceWarning: lb fgs failed to converge (status=1):

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:

https://scikit-learn.org/stable/modules/preprocessing.html

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression

n\_iter\_i = \_check\_optimize\_result(

C:\Users\alpika.gupta\Anaconda3\lib\site-packages\sklearn\linear\_model\\_logistic.py:444: ConvergenceWarning: lb fgs failed to converge (status=1):

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:

https://scikit-learn.org/stable/modules/preprocessing.html

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression

n\_iter\_i = \_check\_optimize\_result(

C:\Users\alpika.gupta\Anaconda3\lib\site-packages\sklearn\linear\_model\\_logistic.py:444: ConvergenceWarning: lb fgs failed to converge (status=1):

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:

https://scikit-learn.org/stable/modules/preprocessing.html

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression

n\_iter\_i = \_check\_optimize\_result(

0.7668831168831169

C:\Users\alpika.gupta\Anaconda3\lib\site-packages\sklearn\linear\_model\\_logistic.py:444: ConvergenceWarning: lb fgs failed to converge (status=1):

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:

https://scikit-learn.org/stable/modules/preprocessing.html

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression

n\_iter\_i = \_check\_optimize\_result(

C:\Users\alpika.gupta\Anaconda3\lib\site-packages\sklearn\linear\_model\\_logistic.py:444: ConvergenceWarning: lb fgs failed to converge (status=1):

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:

https://scikit-learn.org/stable/modules/preprocessing.html

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression

n\_iter\_i = \_check\_optimize\_result(

**Note: In this lesson, we saw the use of the ensemble learning methods, and in the next lesson, we will be working on Recommender Systems.**

****