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

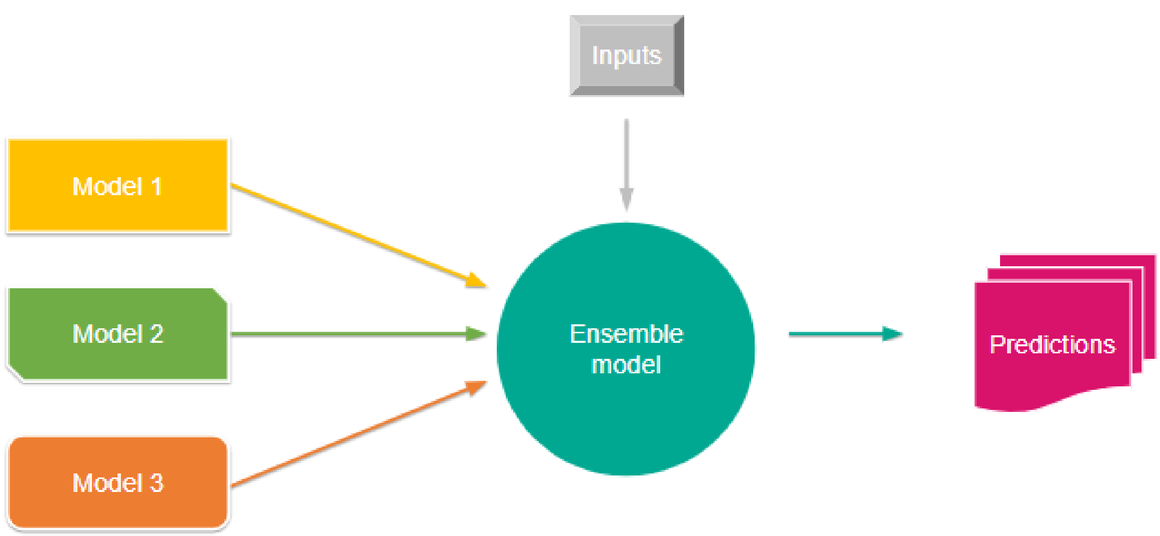
What Is Ensemble Learning?

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

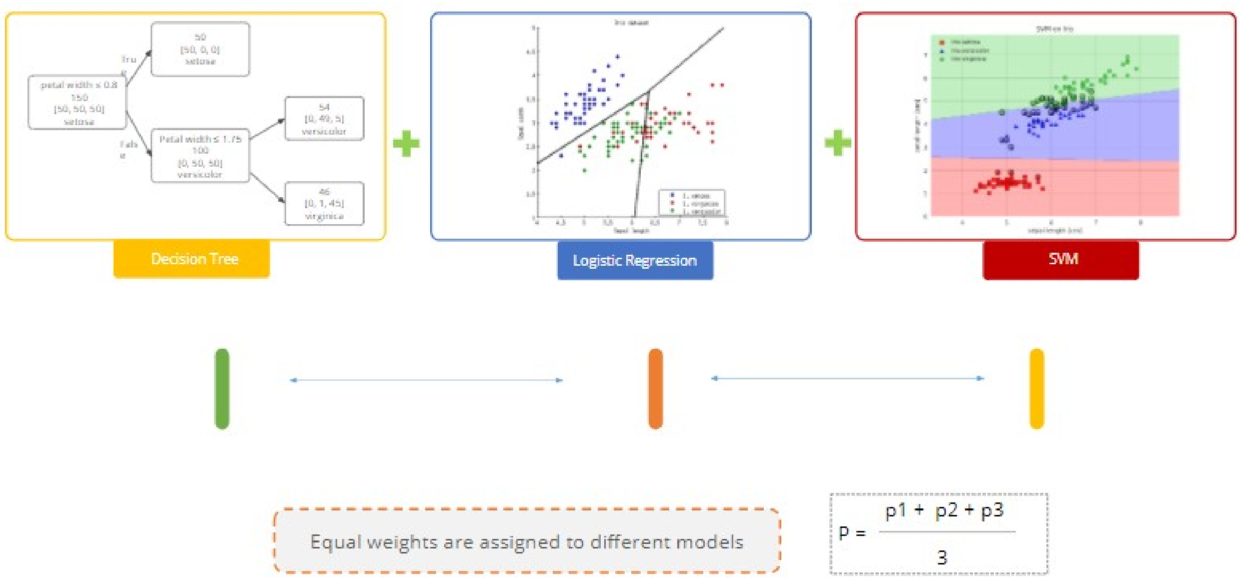
Every ensemble algorithm consists of two steps:

- Producing a cohort of predictions using simple ML algorithms
- Combining the predictions into one aggregated model

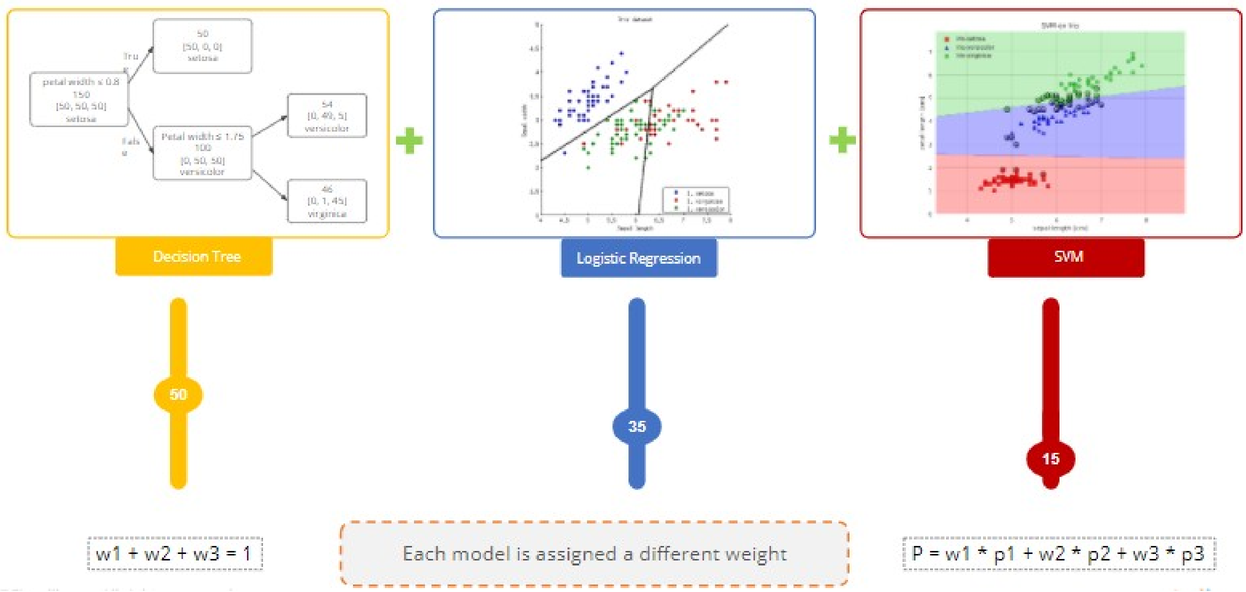
The ensemble can be achieved through several techniques.

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.

```
#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.

```
#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())

0.7721975393028024
```

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.

```
#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())

0.7670027341079972
```

Boosting Algorithms

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

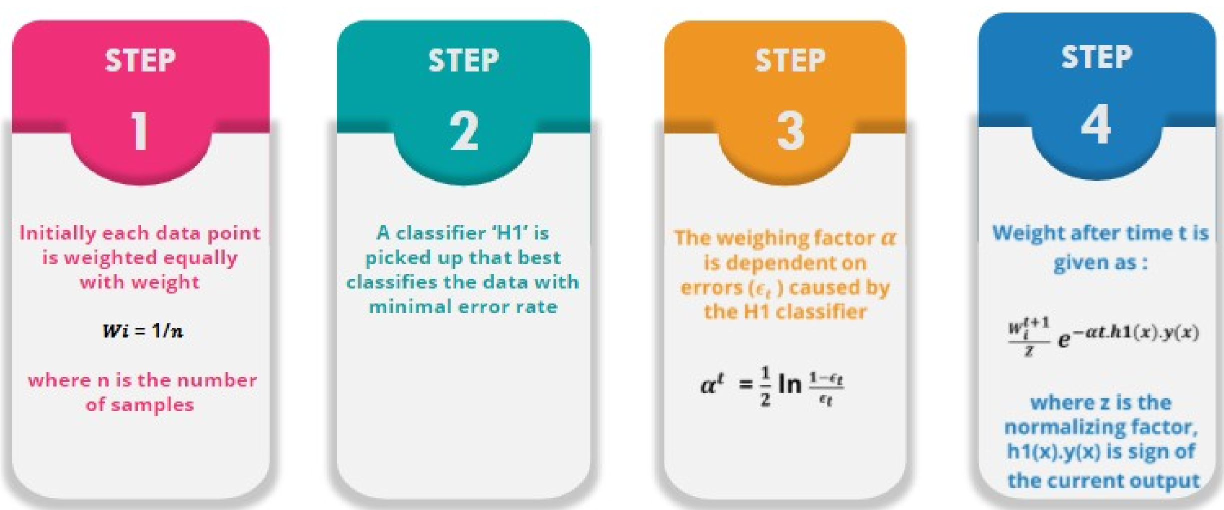
Once created, the models make predictions that may be weighted by their demonstrated accuracy, and the results are combined to create a final output prediction.

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.

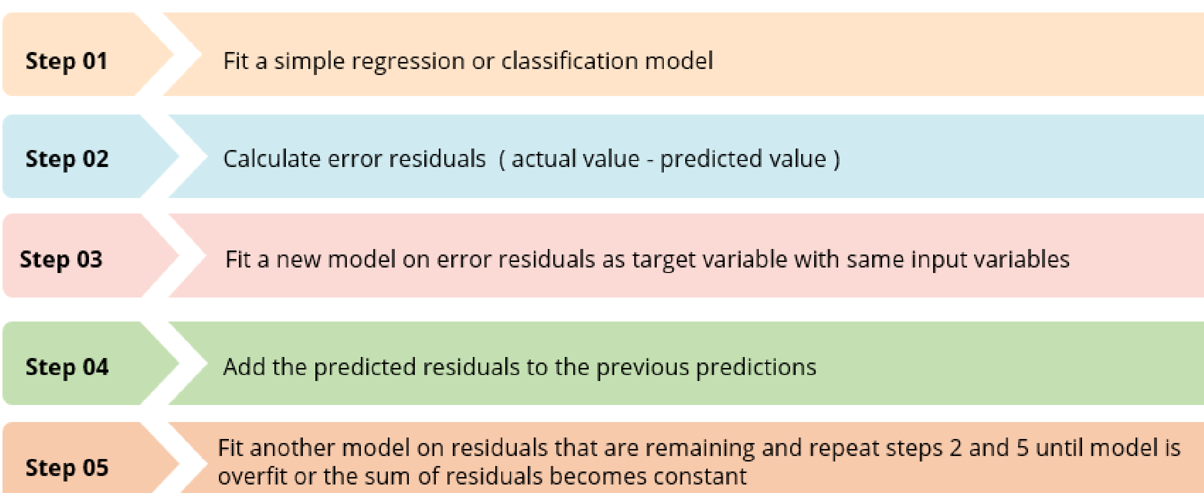
```
#AdaBoost Classification
import pandas
from sklearn import model_selection
from sklearn.ensemble import AdaBoostClassifier
```

```
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.

```
#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())

0.7604921394395079
```

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

```
#Installing CatBoost
pip install catboost

#To import libraries
import catboost
print(catboost.__version__)
!python --version

1.0.4
Python 2.7.17
```

Reading the Data

```
#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

(train_df, test_df) = catboost.datasets.amazon()

train_df.head()

   ACTION  RESOURCE  MGR_ID  ROLE_ROLLUP_1  ROLE_ROLLUP_2
ROLE_DEPTNAME  \
0       1     39353   85475         117961         118300
123472
1       1     17183    1540         117961         118343
123125
2       1     36724   14457         118219         118220
117884
3       1     36135    5396         117961         118343
119993
4       1     42680    5905         117929         117930
```

```
119569

   ROLE_TITLE  ROLE_FAMILY_DESC  ROLE_FAMILY  ROLE_CODE
0      117905            117906       290919     117908
1      118536            118536       308574     118539
2      117879            267952        19721     117880
3      118321            240983       290919     118322
4      119323            123932        19793     119325
```

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.

```
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.

```
#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]

#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.

```
#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
)

!head amazon/train.csv

ACTION,RESOURCE,MGR_ID,ROLE_ROLLUP_1,ROLE_ROLLUP_2,ROLE_DEPTNAME,ROLE_
TITLE,ROLE_FAMILY_DESC,ROLE_FAMILY,ROLE_CODE
1,39353,85475,117961,118300,123472,117905,117906,290919,117908
1,17183,1540,117961,118343,123125,118536,118536,308574,118539
1,36724,14457,118219,118220,117884,117879,267952,19721,117880
1,36135,5396,117961,118343,119993,118321,240983,290919,118322
1,42680,5905,117929,117930,119569,119323,123932,19793,119325
0,45333,14561,117951,117952,118008,118568,118568,19721,118570
1,25993,17227,117961,118343,123476,118980,301534,118295,118982
1,19666,4209,117961,117969,118910,126820,269034,118638,126822
1,31246,783,117961,118413,120584,128230,302830,4673,128231

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')
)
```

```
!cat amazon/train.cd
```

```
0	Label
1	Categ	RESOURCE
2	Categ	MGR_ID
3	Categ	ROLE_ROLLUP_1
4	Categ	ROLE_ROLLUP_2
5	Categ	ROLE_DEPTNAME
6	Categ	ROLE_TITLE
7	Categ	ROLE_FAMILY_DESC
8	Categ	ROLE_FAMILY
9	Categ	ROLE_CODE
```

```
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', 'ROLE_FAMILY',
'ROLE_CODE']

dataset 2:
['RESOURCE', 'MGR_ID', 'ROLE_ROLLUP_1', 'ROLE_ROLLUP_2',
'ROLE_DEPTNAME', 'ROLE_TITLE', 'ROLE_FAMILY_DESC', 'ROLE_FAMILY',
'ROLE_CODE']

dataset 3:
['RESOURCE', 'MGR_ID', 'ROLE_ROLLUP_1', 'ROLE_ROLLUP_2',
'ROLE_DEPTNAME', 'ROLE_TITLE', 'ROLE_FAMILY_DESC', 'ROLE_FAMILY',
'ROLE_CODE']

dataset 4:
['RESOURCE', 'MGR_ID', 'ROLE_ROLLUP_1', 'ROLE_ROLLUP_2',
'ROLE_DEPTNAME', 'ROLE_TITLE', 'ROLE_FAMILY_DESC', 'ROLE_FAMILY',
'ROLE_CODE']
```

Split Your Data into Train and Validation

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

```
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.

```python
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.

```python
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:
10.2ms	remaining: 143ms
1:	learn: 0.3149660	test: 0.3151186	best: 0.3151186 (1)	total:
21.6ms	remaining: 141ms
2:	learn: 0.2621494	test: 0.2629766	best: 0.2629766 (2)	total:
30.6ms	remaining: 123ms
3:	learn: 0.2302316	test: 0.2302315	best: 0.2302315 (3)	total:
41.5ms	remaining: 114ms
4:	learn: 0.2060274	test: 0.2019603	best: 0.2019603 (4)	total:
50.6ms	remaining: 101ms
5:	learn: 0.1956107	test: 0.1894627	best: 0.1894627 (5)	total:
59.3ms	remaining: 89ms
6:	learn: 0.1870345	test: 0.1790904	best: 0.1790904 (6)	total:
```

```
69.2ms      remaining: 79.1ms
7:    learn: 0.1836943 test: 0.1748030  best: 0.1748030 (7)   total:
78.1ms      remaining: 68.3ms
8:    learn: 0.1807119 test: 0.1707896  best: 0.1707896 (8)   total:
86.5ms      remaining: 57.7ms
9:    learn: 0.1775777 test: 0.1662489  best: 0.1662489 (9)   total:
96ms  remaining: 48ms
10:   learn: 0.1762130 test: 0.1654446  best: 0.1654446 (10)  total:
105ms remaining: 38.1ms
11:   learn: 0.1760650 test: 0.1653191  best: 0.1653191 (11)  total:
109ms remaining: 27.3ms
12:   learn: 0.1748232 test: 0.1642093  best: 0.1642093 (12)  total:
118ms remaining: 18.1ms
13:   learn: 0.1742028 test: 0.1638902  best: 0.1638902 (13)  total:
127ms remaining: 9.05ms
14:   learn: 0.1733966 test: 0.1627237  best: 0.1627237 (14)  total:
135ms remaining: 0us

bestTest = 0.162723674
bestIteration = 14


<catboost.core.CatBoostClassifier at 0x7f8d887807d0>
```

Metric Calculation and Graph Plotting

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

```
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
)

{"model_id":"d4a874c1c53a4e838f4c23ac98f85d24","version_major":2,"vers
ion_minor":0}

<catboost.core.CatBoostClassifier at 0x7f8d8870d8d0>
```

Model Comparison

Let us compare the models.

```
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
)
```

```
<catboost.core.CatBoostClassifier at 0x7f8d891f3c50>
```

```
from catboost import MetricVisualizer
MetricVisualizer(['learing_rate_0.01', 'learing_rate_0.7']).start()
```

{"model_id":"82f6f10a397b468f8acb7e26d289ed49","version_major":2,"version_minor":0}

Best Iteration

```
#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
)
```

{"model_id":"a0a59673a8cb47ec95ee513789c0a61c","version_major":2,"version_minor":0}

```
<catboost.core.CatBoostClassifier at 0x7f8d8870d310>
```

```
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.

```
#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
)
```

{"model_id":"3ff3b0e279cd4b399718e8947d61ceb1","version_major":2,"version_minor":0}

```
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 = 0.15425211
bestIteration = 35

Training on fold [3/5]

bestTest = 0.1433537051
bestIteration = 55

Training on fold [4/5]

bestTest = 0.1560519524
bestIteration = 55
```

```
cv_data.head()
```

```
   iterations  test-Logloss-mean  test-Logloss-std  train-Logloss-mean  \
0           0           0.302367          0.004317            0.302196

1           1           0.227370          0.007679            0.228497

2           2           0.190856          0.006917            0.196796

3           3           0.178884          0.007455            0.186682

4           4           0.172286          0.007957            0.181380


   train-Logloss-std  test-AUC-mean  test-AUC-std
0           0.004517       0.513577      0.030360
1           0.005126       0.642263      0.048004
2           0.003999       0.791709      0.011361
3           0.003242       0.813889      0.009362
4           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**.

```
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: 0.1582±0.0102 on step
53
```

```
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)
)
```

{"model_id":"b5b4aa4b383e46e998644f18996bced8","version_major":2,"vers
ion_minor":0}

```
Training on fold [0/5]

bestTest = 0.1614486451
bestIteration = 31

Training on fold [1/5]

bestTest = 0.1551886688
bestIteration = 56

Training on fold [2/5]

bestTest = 0.1597838545
bestIteration = 25

Training on fold [3/5]

bestTest = 0.1523066165
bestIteration = 56
```

```
Training on fold [4/5]

bestTest = 0.1577738401
bestIteration = 30

Best validation Logloss score, stratified: 0.1580±0.0041 on step 56
```

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.

```
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
)
```

{"model_id":"9eec2a0b54904c8b88a573d249aab21a","version_major":2,"version_minor":0}

```
<catboost.core.CatBoostClassifier at 0x7f8d886fd690>
```

```
print(model_with_early_stop.tree_count_)
```

```
30
```

```
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
)
```

{"model_id":"7a7827852e6e4d4dae09925aa203ed7a","version_major":2,"version_minor":0}

```
<catboost.core.CatBoostClassifier at 0x7f8d8860f810>
```

```
print(model_with_early_stop.tree_count_)
```

```
30
```

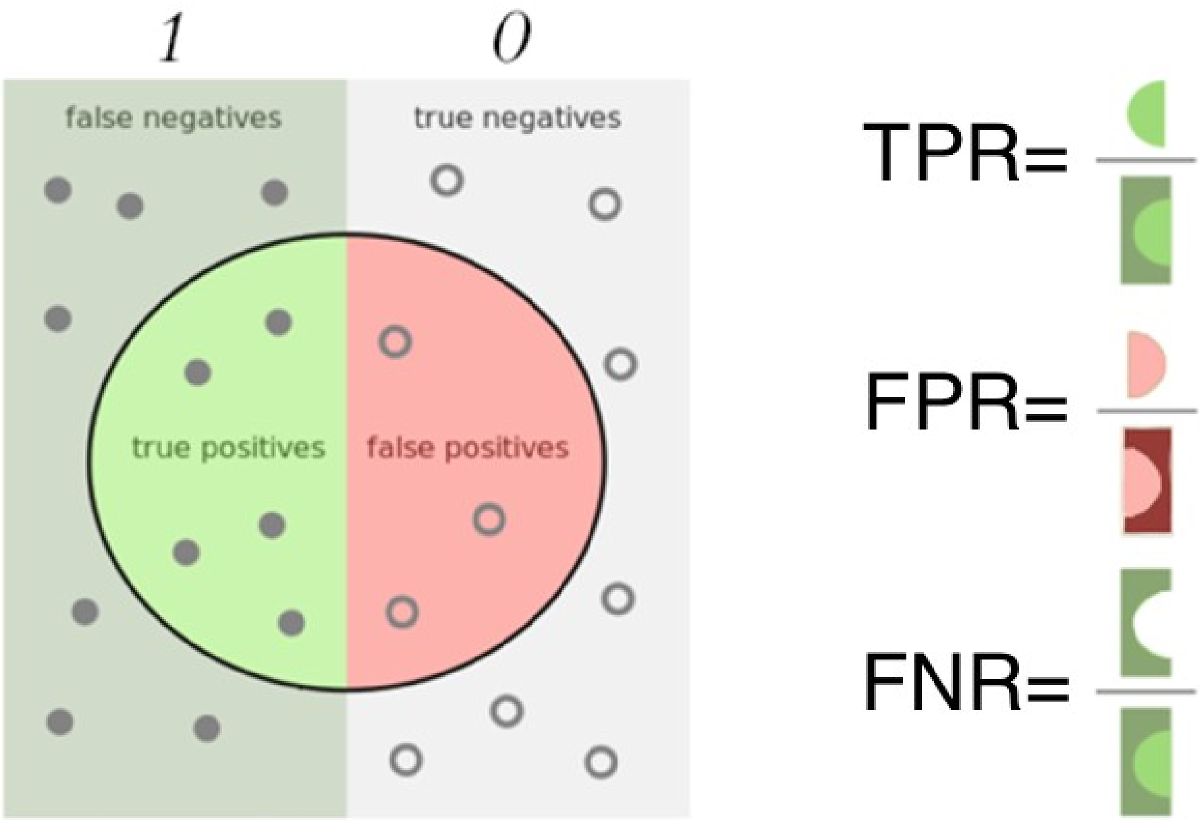
Select Decision Boundary

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.

```
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
)
```

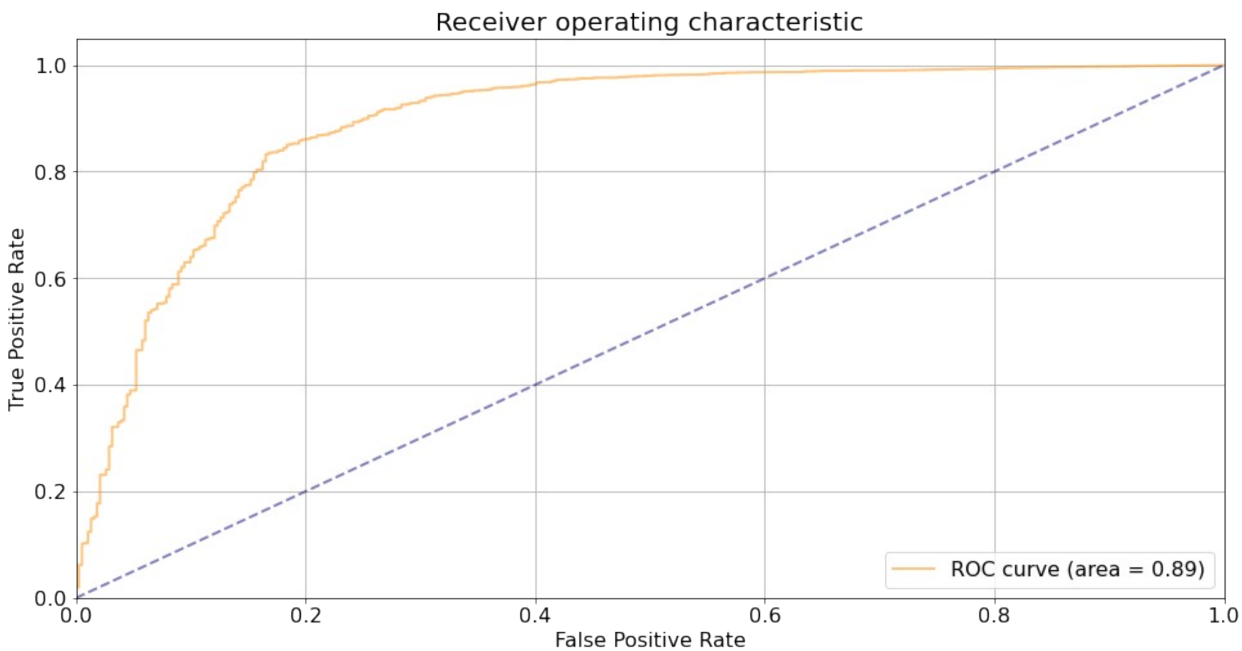
{"model_id":"a78e54693e1148d4960144fdbf0ffe97","version_major":2,"version_minor":0}

```
<catboost.core.CatBoostClassifier at 0x7f8dcd70f150>
```




```
#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)

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])
```

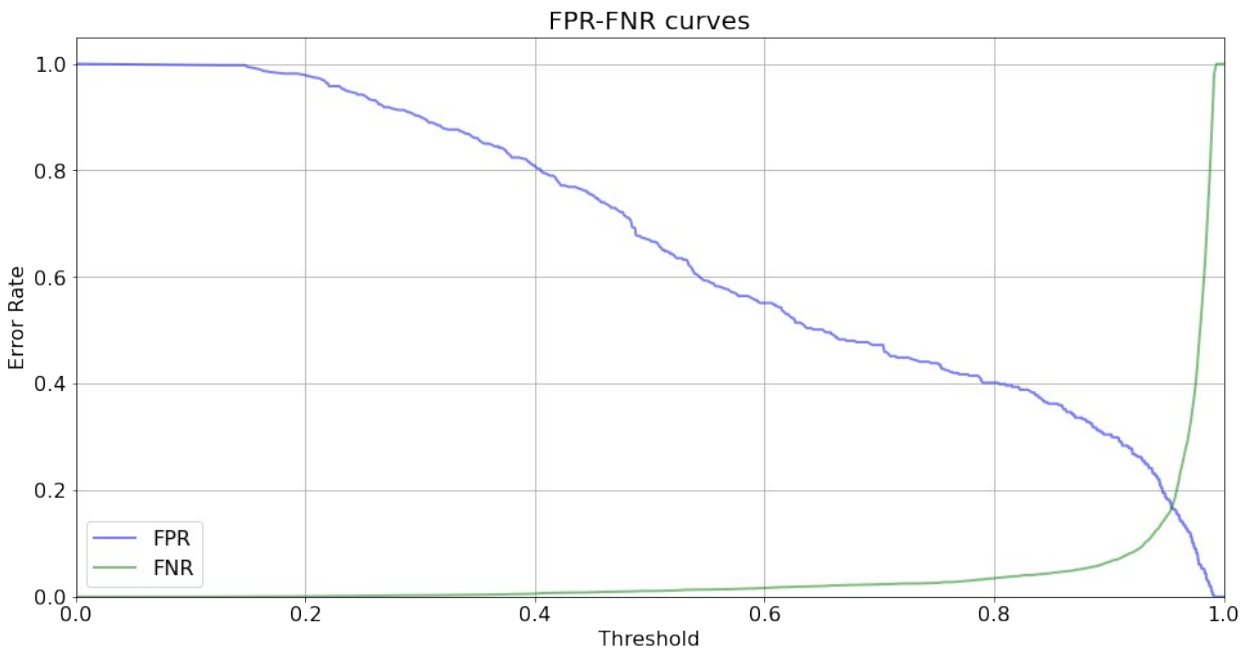
```
plt.ylim([0.0, 1.05])
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.grid(True)
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.title('Receiver operating characteristic', fontsize=20)
plt.legend(loc="lower right", fontsize=16)
plt.show()
```



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

```
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)

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.

```
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))

0.4805444481363058
0.9900857295557712
```

Snapshotting

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

```
#!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

bestTest = 0.1575677776
bestIteration = 80

Shrink model to first 81 iterations.

<catboost.core.CatBoostClassifier at 0x7f8d783bc0d0>
```

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.

```
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]]
```

```
print(model.predict(data=X_validation))
```

```
[1 1 1 ... 1 1 1]
```

```
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]
```

```
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.

```
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).

```
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.**

```
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
)
```

{"model_id":"22b5a0cdd5e649d9b35a4a039fb52bd1","version_major":2,"version_minor":0}

```
<catboost.core.CatBoostClassifier at 0x7f8d78460790>
```

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.

```
#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)

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
)
```


```
{"model_id":"52b87a3ce3684f7e958ab1df8380d9d7","version_major":2,"vers
ion_minor":0}
```


```
<catboost.core.CatBoost at 0x7f8d88618f90>
```

Making predictions with ranking mode

```
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.

```
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: 16.7ms	remaining: 3.32s
50:	learn: 0.1923495	total: 1.06s	remaining: 3.1s
100:	learn: 0.1653594	total: 2.3s	remaining: 2.25s
150:	learn: 0.1570631	total: 3.82s	remaining: 1.24s
199:	learn: 0.1538962	total: 5.31s	remaining: 0us
```

```
<catboost.core.CatBoostClassifier at 0x7f8d783f8710>
```

```
metrics = model.eval_metrics(
    data=pool1,
    metrics=['Logloss','AUC'],
    ntree_start=0,
    ntree_end=0,
    eval_period=1,
    plot=True
)
```

{"model_id":"22b276b56dda46f999aec525cf70dfb1","version_major":2,"vers
ion_minor":0}

```
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 0.7731 0.7769 0.7866 0.7985 0.7986 0.8008 0.8004 0.8004
 0.8191 0.8357 0.8518 0.8666 0.8851 0.8855 0.8886 0.8931 0.8936 0.8991
 0.9033 0.9115 0.9126 0.9136 0.9148 0.9163 0.9177 0.9184 0.9206 0.9211
 0.9259 0.9289 0.9291 0.9324 0.9329 0.9334 0.9338 0.9358 0.937  0.9383
 0.9386 0.9385 0.939  0.9396 0.94   0.9401 0.941  0.9411 0.942  0.9425
 0.944  0.9457 0.9471 0.9479 0.9489 0.9499 0.9512 0.9522 0.9527 0.9533
 0.9537 0.9541 0.9543 0.9547 0.955  0.9553 0.9554 0.9558 0.9558 0.9563
 0.9575 0.9584 0.9592 0.9597 0.9603 0.961  0.9614 0.9617 0.962  0.9624
 0.9627 0.963  0.9634 0.964  0.9642 0.9644 0.9648 0.9649 0.9653 0.9655
 0.9657 0.9657 0.9658 0.966  0.9661 0.9662 0.9663 0.9665 0.9665 0.9666
 0.9667 0.9669 0.967  0.9675 0.968  0.9683 0.9689 0.9694 0.9699 0.9703
 0.9706 0.9709 0.9711 0.9715 0.9715 0.9718 0.9719 0.9719 0.9723 0.9723
 0.9725 0.9726 0.9728 0.973  0.9732 0.9734 0.9734 0.9734 0.9736 0.9736
 0.9741 0.9741 0.9745 0.975  0.975  0.9752 0.9752 0.9752 0.9754 0.9758
 0.9761 0.9761 0.9762 0.9763 0.9763 0.9763 0.9763 0.9763 0.9763 0.9763
 0.9763 0.9765 0.9765 0.9768 0.9769 0.9771 0.9772 0.9774 0.9777 0.9778
 0.9778 0.9779 0.978  0.978  0.9781 0.9781 0.9781 0.9783 0.9786 0.9786
 0.9787 0.9787 0.9787 0.9789 0.9789 0.979  0.979  0.979  0.979
0.979 ]
```

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.

```
#To find feature importance
model.get_feature_importance(prettified=True)
```

	Feature Id	Importances
0	RESOURCE	22.459777
1	MGR_ID	17.115632
2	ROLE_DEPTNAME	16.054805
3	ROLE_ROLLUP_2	13.975879
4	ROLE_CODE	10.030076
5	ROLE_FAMILY_DESC	7.517166
6	ROLE_TITLE	6.526255
7	ROLE_FAMILY	3.704980
8	ROLE_ROLLUP_1	2.615430

Scores are assigned to the input features.

Feature Evaluation

Let us perform feature evaluation using the **eval_features( )** function.

```
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])

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)
)
```

```
                  PValue     Score  Quantile 0.005  Quantile 0.995
Decision
Features: 6  0.000189  1.010962        0.582841        1.449307
GOOD
Features: 7  0.681322 -0.033237       -0.331248        0.284845
UNKNOWN
Features: 8  0.005111 -0.439271       -0.761790       -0.091166
BAD
```

Saving the Model

```
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')

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

```
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
)
```

{"model_id":"76030f163f7c44199cbae12b33a3b1e3","version_major":2,"version_minor":0}

```
<catboost.core.CatBoostClassifier at 0x7f8d7a22e9d0>
```

Accuracy

```
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
)
```

{"model_id":"ef0114e102ed4ff694f884264ec7d64d","version_major":2,"version_minor":0}

```
<catboost.core.CatBoostClassifier at 0x7f8d79d5b6d0>
```

Training the Model after Parameter Tuning

```
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 0.043372
0:    learn: 0.6422041 total: 16ms      remaining: 16.9s
100:  learn: 0.1537302 total: 2.94s     remaining: 27.8s
```

```
200:	learn: 0.1466783	total: 6.66s	remaining: 28.4s
300:	learn: 0.1428331	total: 10.3s	remaining: 26s
400:	learn: 0.1389527	total: 14.2s	remaining: 23.3s
500:	learn: 0.1354927	total: 18.1s	remaining: 20.1s
600:	learn: 0.1327491	total: 22s	remaining: 16.7s
700:	learn: 0.1297104	total: 25.9s	remaining: 13.2s
800:	learn: 0.1270650	total: 29.7s	remaining: 9.57s
900:	learn: 0.1243689	total: 33.5s	remaining: 5.88s
1000:	learn: 0.1221292	total: 37.6s	remaining: 2.18s
1058:	learn: 0.1207668	total: 39.8s	remaining: 0us

<catboost.core.CatBoostClassifier at 0x7f8d79d6ad50>
```

Calculate Prediction

```
#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:
[[0.4144 0.5856]
 [0.0167 0.9833]
 [0.0102 0.9898]
 ...
 [0.0052 0.9948]
 [0.0438 0.9562]
 [0.0113 0.9887]]
```

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.

```python
#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())
```

```
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/
_logistic.py:765: ConvergenceWarning: lbfgs 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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/_logistic.
py:765: ConvergenceWarning: lbfgs 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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/_logistic.
py:765: ConvergenceWarning: lbfgs 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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/_logistic.
py:765: ConvergenceWarning: lbfgs 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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/_logistic.
py:765: ConvergenceWarning: lbfgs 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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/_logistic.
py:765: ConvergenceWarning: lbfgs 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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/_logistic.
py:765: ConvergenceWarning: lbfgs 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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/_logistic.
py:765: ConvergenceWarning: lbfgs 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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)

0.76296992481203

/usr/local/lib/python3.7/site-packages/sklearn/linear_model/
_logistic.py:765: ConvergenceWarning: lbfgs 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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/_logistic.
py:765: ConvergenceWarning: lbfgs 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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

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.