XGBoost with Python and Scikit-Learn

XGBoost is an acronym for **Extreme Gradient Boosting**. It is a powerful machine learning algorithm that can be used to solve classification and regression problems. In this project, I implement XGBoost with Python and Scikit-Learn to solve a classification problem. The problem is to classify the customers from two different channels as Horeca (Hotel/Retail/Café) customers or Retail channel (nominal) customers.

Table of Contents

1. Introduction to XGBoost algorithm

XGBoost stands for **Extreme Gradient Boosting**. XGBoost is a powerful machine learning algorithm that is dominating the world of applied machine learning and Kaggle competitions. It is an implementation of gradient boosted trees designed for speed and accuracy.

XGBoost (Extreme Gradient Boosting) is an advanced implementation of the gradient boosting algorithm. It has proved to be a highly effective machine learning algorithm extensively used in machine learning competitions. XGBoost has high predictive power and is almost 10 times faster than other gradient boosting techniques. It also includes a variety of regularization parameters which reduces overfitting and improves overall performance. Hence, it is also known as **regularized boosting** technique.

2. XGBoost algorithm intuition

XGBoost (Extreme Gradient Boosting) belongs to a family of boosting algorithms. It uses the gradient boosting (GBM) framework at its core. So, first of all we should know about gradient boosting.

Gradient boosting

Gradient boosting is a supervised machine learning algorithm, which tries to predict a target variable by combining the estimates of a set of simpler, weaker models. In boosting, the trees are built in a sequential manner such that each subsequent tree aims to reduce the errors of the previous tree. The misclassified labels are given higher weights. Each tree learns from its predecessors and tries to reduce the residual errors. So, the tree next in sequence will learn from the previous tree residuals.

XGBoost

In XGBoost, we try to fit a model on the gradient of the loss function generated from the previous step. So, in XGBoost we modified our gradient boosting algorithm so that it works with any differentiable loss function.

3. The problem statement

In this project, I try to solve a classification problem. The problem is to classify the customers from two different channels as Horeca (Hotel/Retail/Café) customers or Retail channel (nominal) customers. I implement XGBoost with Python and Scikit-Learn to solve the classification problem.

4. Dataset description

I have used the `Wholesale customers data set` for this project, downloaded from the UCI Machine learning repository. This dataset can be found at the following url-

https://archive.ics.uci.edu/ml/datasets/Wholesale+customers

5. Import libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import warnings

warnings.filterwarnings('ignore')
```

6. Import dataset

```
# Import dataset

data = 'C:/datasets/Wholesale customers data.csv'

df = pd.read_csv(data)
```

7. Exploratory Data Analysis

I will start off by checking the shape of the dataset.

```
df.shape
```

```
(440, 8)
```

We can see that there are 440 instances and 8 attributes in the dataset.

Preview dataset

```
df.head()
```

```
   Channel  Region  Fresh  Milk  Grocery  Frozen  Detergents_Paper
Delicassen
0        2       3  12669  9656     7561     214              2674
1338
1        2       3   7057  9810     9568    1762              3293
1776
2        2       3   6353  8808     7684    2405              3516
7844
3        1       3  13265  1196     4221    6404               507
1788
4        2       3  22615  5410     7198    3915              1777
5185
```

We can see that `Channel` variable contains values as `1` and `2`. These two values classify the customers from two different channels as 1 for Horeca (Hotel/Retail/Café) customers and 2 for Retail channel (nominal) customers.

View summary of dataframe

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 440 entries, 0 to 439
Data columns (total 8 columns):
Channel             440 non-null int64
Region              440 non-null int64
Fresh               440 non-null int64
Milk                440 non-null int64
Grocery             440 non-null int64
Frozen              440 non-null int64
Detergents_Paper    440 non-null int64
Delicassen          440 non-null int64
dtypes: int64(8)
memory usage: 27.6 KB
```

We can see that there are only numerical variables in the dataset.

View summary statistics of dataframe

```
df.describe()
```

```
          Channel      Region          Fresh          Milk
Grocery  \
count  440.000000  440.000000     440.000000    440.000000
440.000000
mean     1.322727    2.543182   12000.297727   5796.265909
7951.277273
std      0.468052    0.774272   12647.328865   7380.377175
9503.162829
min      1.000000    1.000000       3.000000     55.000000
3.000000
25%      1.000000    2.000000    3127.750000   1533.000000
2153.000000
50%      1.000000    3.000000    8504.000000   3627.000000
4755.500000
75%      2.000000    3.000000   16933.750000   7190.250000
10655.750000
max      2.000000    3.000000  112151.000000  73498.000000
92780.000000

             Frozen  Detergents_Paper    Delicassen
count    440.000000        440.000000    440.000000
mean    3071.931818       2881.493182   1524.870455
std     4854.673333       4767.854448   2820.105937
min       25.000000          3.000000      3.000000
25%      742.250000        256.750000    408.250000
50%     1526.000000        816.500000    965.500000
75%     3554.250000       3922.000000   1820.250000
max    60869.000000      40827.000000  47943.000000
```

Check for missing values

```
df.isnull().sum()
```

```
Channel             0
Region              0
Fresh               0
Milk                0
Grocery             0
Frozen              0
Detergents_Paper    0
Delicassen          0
dtype: int64
```

There are no missing values in the dataset.

8. Declare feature vector and target variable

```
X = df.drop('Channel', axis=1)

y = df['Channel']
```

let's take a look at feature vector(X) and target variable(y)

```
X.head()
```

	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	3	12669	9656	7561	214	2674	1338
1	3	7057	9810	9568	1762	3293	1776
2	3	6353	8808	7684	2405	3516	7844
3	3	13265	1196	4221	6404	507	1788
4	3	22615	5410	7198	3915	1777	5185

```
y.head()
```

```
0    2
1    2
2    2
3    1
4    2
Name: Channel, dtype: int64
```

We can see that the y labels contain values as 1 and 2. I will need to convert it into 0 and 1 for further analysis. I will do it as follows-

```
# convert labels into binary values

y[y == 2] = 0

y[y == 1] = 1

# again preview the y label

y.head()
```

```
0    0
1    0
2    0
3    1
4    0
Name: Channel, dtype: int64
```

Now, I will convert the dataset into an optimized data structure called **Dmatrix** that XGBoost supports and gives it acclaimed performance and efficiency gains. I will do it as follows.

```
# import XGBoost
import xgboost as xgb


# define data_dmatrix
data_dmatrix = xgb.DMatrix(data=X,label=y)
```

9. Split data into separate training and test set

```
# split X and y into training and testing sets

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.3, random_state = 0)
```

10. Train the XGBoost classifier

- Now, I will train the XGBoost classifier. We need to know different parameters that XGBoost provides. There are three types of parameters that we must set before running XGBoost. These parameters are as follows:-

General parameters

These parameters relate to which booster we are doing boosting. The common ones are tree or linear model.

Booster parameters

It depends on which booster we have chosen for boosting.

Learning task parameters

These parameters decide on the learning scenario. For example, regression tasks may use different parameters than ranking tasks.

Command line parameters

In addition there are command line parameters which relate to behaviour of CLI version of XGBoost.

The most important parameters that we should know about are as follows:-

learning_rate - It gives us the step size shrinkage which is used to prevent overfitting. Its range is [0,1].

max_depth - It determines how deeply each tree is allowed to grow during any boosting round.

subsample - It determines the percentage of samples used per tree. Low value of subsample can lead to underfitting.

colsample_bytree - It determines the percentage of features used per tree. High value of it can lead to overfitting.

n_estimators - It is the number of trees we want to build.

objective - It determines the loss function to be used in the process. For example, `reg:linear` for regression problems, `reg:logistic` for classification problems with only decision, `binary:logistic` for classification problems with probability.

XGBoost also supports regularization parameters to penalize models as they become more complex and reduce them to simple models. These regularization parameters are as follows:-

gamma - It controls whether a given node will split based on the expected reduction in loss after the split. A higher value leads to fewer splits. It is supported only for tree-based learners.

alpha - It gives us the `L1` regularization on leaf weights. A large value of it leads to more regularization.

lambda - It gives us the `L2` regularization on leaf weights and is smoother than `L1` regularization.

Though we are using trees as our base learners, we can also use XGBoost's relatively less popular linear base learners and one other tree learner known as `dart`. We have to set the `booster` parameter to either `gbtree` (default), `gblinear` or `dart`.

```
# import XGBClassifier
from xgboost import XGBClassifier


# declare parameters
params = {
            'objective':'binary:logistic',
            'max_depth': 4,
            'alpha': 10,
            'learning_rate': 1.0,
            'n_estimators':100
        }



# instantiate the classifier
xgb_clf = XGBClassifier(**params)



# fit the classifier to the training data
xgb_clf.fit(X_train, y_train)
```

```
XGBClassifier(alpha=10, base_score=0.5, booster='gbtree',
colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0,
learning_rate=1.0,
```

```
       max_delta_step=0, max_depth=4, min_child_weight=1,
missing=None,
       n_estimators=100, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
       subsample=1, verbosity=1)
```

```
# alternatively view the parameters of the xgb trained model
print(xgb_clf)
```

```
XGBClassifier(alpha=10, base_score=0.5, booster='gbtree',
colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0,
learning_rate=1.0,
       max_delta_step=0, max_depth=4, min_child_weight=1,
missing=None,
       n_estimators=100, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
       subsample=1, verbosity=1)
```

11. Make predictions with XGBoost Classifier

```
# make predictions on test data
y_pred = xgb_clf.predict(X_test)
```

12. Check accuracy score

```
# check accuracy score
from sklearn.metrics import accuracy_score

print('XGBoost model accuracy score: {0:0.4f}'.
format(accuracy_score(y_test, y_pred)))
```

```
XGBoost model accuracy score: 0.9167
```

We can see that XGBoost obtain very high accuracy score of 91.67%.

13. k-fold Cross Validation using XGBoost

To build more robust models with XGBoost, we must do k-fold cross validation. In this way, we ensure that the original training dataset is used for both training and validation. Also, each entry is used for validation just once. XGBoost supports k-fold cross validation using the `cv()` method. In this method, we will specify several parameters which are as follows:-

nfolds - This parameter specifies the number of cross-validation sets we want to build.

num_boost_round - It denotes the number of trees we build.

metrics - It is the performance evaluation metrics to be considered during CV.

as_pandas - It is used to return the results in a pandas DataFrame.

early_stopping_rounds - This parameter stops training of the model early if the hold-out metric does not improve for a given number of rounds.

seed - This parameter is used for reproducibility of results.

We can use these parameters to build a k-fold cross-validation model by calling `XGBoost's CV()` method.

```
from xgboost import cv

params = {"objective":"binary:logistic",'colsample_bytree':
0.3,'learning_rate': 0.1,
                'max_depth': 5, 'alpha': 10}

xgb_cv = cv(dtrain=data_dmatrix, params=params, nfold=3,
                    num_boost_round=50, early_stopping_rounds=10,
metrics="auc", as_pandas=True, seed=123)
```

`xgb_cv` contains train and test `auc` metrics for each boosting round. Let's preview `xgb_cv`.

```
xgb_cv.head()
```

	train-auc-mean	train-auc-std	test-auc-mean	test-auc-std
0	0.914998	0.009704	0.880965	0.021050
1	0.934374	0.013263	0.923561	0.022810
2	0.936252	0.013723	0.924433	0.025777
3	0.943878	0.009032	0.927152	0.022228
4	0.957880	0.008845	0.935191	0.016437

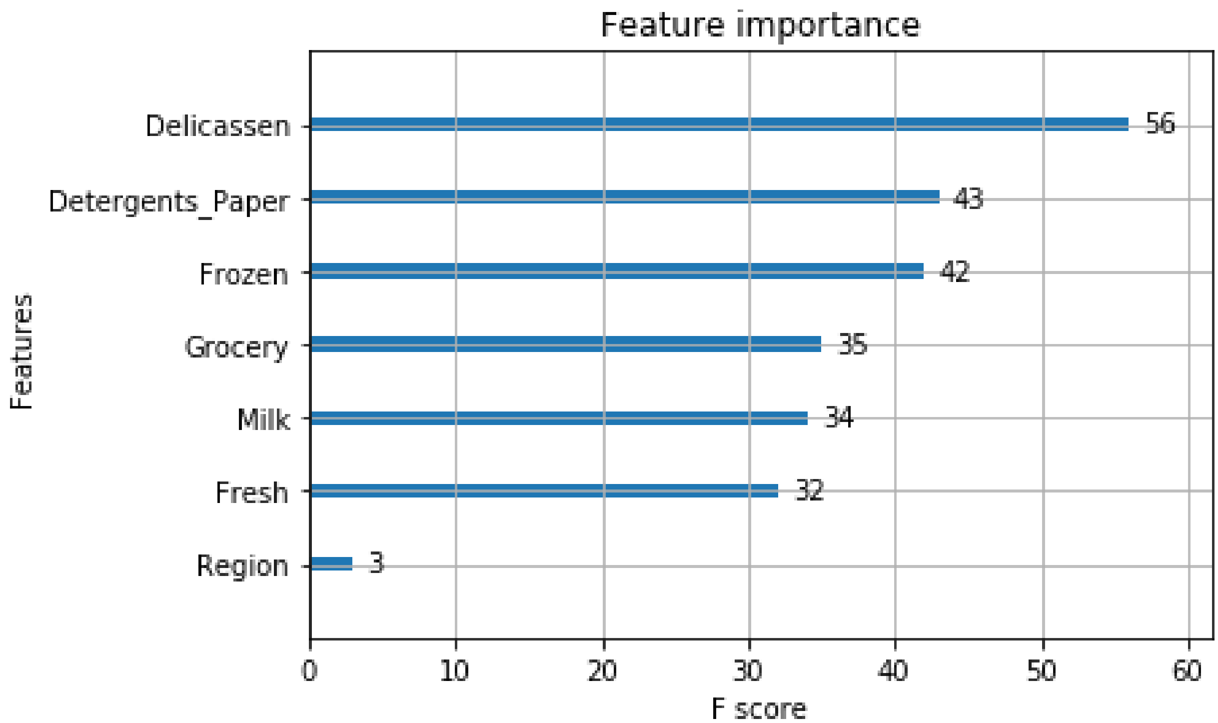
14. Feature importance with XGBoost

XGBoost provides a way to examine the importance of each feature in the original dataset within the model. It involves counting the number of times each feature is split on across all boosting trees in the model. Then we visualize the result as a bar graph, with the features ordered according to how many times they appear.

XGBoost has a **plot_importance()** function that helps us to achieve this task. Then we can visualize the features that has been given the highest important score among all the features. Thus XGBoost provides us a way to do feature selection.

I will proceed as follows:-

```
xgb.plot_importance(xgb_clf)
plt.rcParams['figure.figsize'] = [6, 4]
plt.show()
```



We can see that the feature `Grocery` has been given the highest importance score among all the features. Thus XGBoost also gives us a way to do Feature Selection.

15. Results and conclusion

1. In this project, I implement XGBoost with Python and Scikit-Learn to classify the customers from two different channels as Horeca (Hotel/Retail/Café) customers or Retail channel (nominal) customers.

2. The y labels contain values as 1 and 2. I have converted them into 0 and 1 for further analysis.

3. I have trained the XGBoost classifier and found the accuracy score to be 91.67%.

4. I have done the hyperparameter tuning in XGBoost by doing k-fold cross-validation.

5. I find the most important feature in XGBoost to be `Grocey`. I did it using the **plot_importance()** function in XGBoost that helps us to achieve this task.