Chapter 11 : Part 2

**XGBoost**

Introduction to XGBoost

**11.2.1 XGBoost:** Installation & Preparing the environment

Congratulations for reaching this last section of the Machine-Learning !!! We hope that you feel expert in machine learning and you're very confident in this world and that you are working on some fascinating machine learning projects.

* XGBoost is one of the most ***popular algorithm*** in machine learning that is quite recently popular but still a *very* *powerful* model especially if you work on *large data-sets*.
* It will offer you very *high* *performance* while being *fast* to *execute*.
* It is important to remind that XGBoost is the most powerful implementation of Gradient Boosting in terms of *model* *performance* and *execution* *speed*.
* XGBoost (eXtreme Gradient Boosting) is an open-source software library which provides a *regularizing* *Gradient Boosting* framework for ***C++***, ***Java***, ***Python***, ***R***, ***Julia***, ***Perl***, and ***Scala***.
* This is only going to be an introduction so we will make a simple implementation of XGBoost. But this simple implementation will definitely give you some excellent performance.
* Install XGBoost and integrate it in spider: Visit the Official website: <https://xgboost.ai/>
* *XGBoost* is an optimized distributed *gradient boosting library* designed to be highly efficient, *flexible* and *portable*. It implements *machine learning* algorithms under the *Gradient Boosting framework*. *XGBoost* provides a *parallel tree boosting* (also known as *GBDT*, *GBM*) that solve many *data science problems* in a *fast* and *accurate* way. To install latest version and installation guide visit following:

<https://xgboost.readthedocs.io/en/stable/>

<https://xgboost.readthedocs.io/en/stable/install.html>



* Install in windows:

pip install --user xgboost

C:\Users\SolLaSi>pip install --user xgboost

Collecting xgboost

Downloading xgboost-1.6.1-py3-none-win\_amd64.whl (125.4 MB)

|████████████████████████████████| 125.4 MB 56 kB/s

Requirement already satisfied: numpy in c:\users\sollasi\anaconda3\lib\site-packages (from xgboost) (1.22.3)

Requirement already satisfied: scipy in c:\users\sollasi\anaconda3\lib\site-packages (from xgboost) (1.5.2)

Installing collected packages: xgboost

Successfully installed xgboost-1.6.1

C:\Users\SolLaSi>

* Conda: You may use the Conda packaging manager to install XGBoost:

conda install -c conda-forge py-xgboost

* Conda should be able to detect the existence of a GPU on your machine and install the correct variant of XGBoost. If you run into issues, try indicating the variant explicitly:

# CPU only

conda install -c conda-forge py-xgboost-cpu

# Use NVIDIA GPU

conda install -c conda-forge py-xgboost-gpu

**11.2.2 XGBoost usin Python:** Problem description & Data-preprocessing

* Problem description: Remember this is the Churn Modeling problem where we need to predict the customers of the bank that will leave the bank. We solved it using ANN in Chapter 8: Deep Learning – ANN. **1** for *leave* and **0** for *stay*.
* Where we classify the customers in two classes: those who will *leave the bank* and those who will *not leave the bank*.
* Accuracy using ANN: Remember for this problem we obtained an ***accuracy*** of ***86%*** but that took quite a while because we trained an ***ANN*** with many ***epochs***.
* In this section we're going to apply ***XGBoost*** on this *Churn Modeling Problem* and you're going to see that we will get the *same* *accuracy*. We'll probably get *86% accuracy* but this will *execute* *in no time* like *instantly* compared to ANN.

*[We can not get a higher accuracy anyway because the accuracy is limited by the problem itself in the sense that there isn't a 100% correlation between the informations of the customers and their decision to leave the bank (yes or no). So 86% is very close to the best accuracy we can obtain for this specific business problem.]*

|  |
| --- |
| * Also notice that, this dataset only contains *13 features* so it's not a *large* *dataset*. * It is important to note that even if this was a very large data-set, ***XGBoost*** would be one of the *best model* in terms of *performance*. That is to get a *good* *accuracy* and *execution* *speed*. * So if you are working with a *large data-set* I strongly encourage you to test ***XGBoost***. |

* Data preprocessing: We'll take the pre processing phase from our ANN file.
* Feature scaling: Feature scaling is compulsory for Deep-Learning so we used it in ANN.
* But in ***XGBoost*** feature-scaling is not necessary. Since ***XGBoost*** is a *Gradient Boosting* model with decision trees hence, featuresscaling *is totally* unnecessary.
* That’s one of the very good thing about ***XGBoost*** beside its high performance & high execution speed, you can keep the interpretation of your *problem/data-set* and of the *results* you'll get after building the model.

|  |
| --- |
| * That's why ***XGBoost*** is so popular because it has the three qualities: * High performance * Fast execution speed. * You can keep all the interpretation of your problem and your model (no feature-scaling). |

**Data pre-processing for XGBoost**

# *Library*

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

**import** matplotlib.pyplot **as** pLt

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

# *--------------------------- Part 1 : Data Preprocessing ---------------------------------*

# *Data Extract*

dataSet = **pd.read\_csv**("Churn\_Modelling.csv")

# *X = dataSet.iloc[:, 3:-1].values # this can be used too*

X = dataSet.iloc[:, 3:13].values # *all columns from index 3, excluding 13 indexed column*

y = dataSet.iloc[:, 13].values # *the last column*

# *------------- Encode Categorical Data  -----------*

**from** sklearn.preprocessing **import** LabelEncoder, OneHotEncoder

**from** sklearn.compose **import** ColumnTransformer

#*Encode "Gender" using LabelEncoder."Gender" is in "3rd-column", hence X[:, 2]*

label\_encode = **LabelEncoder**()

# *Following is applicable to numpy array, if we used "X = dataSet.iloc[:, 3:13]"*

# *X = np.array(X) # it is needed if X is not an Arry. i.e. ".values" not applied*

X[:, 2] = **label\_encode.fit\_transform**(X[:, 2])

**print**(X)

# *For a data-set we can still encode it using Columns "key"*

# *X["Gender"] = label\_encode.fit\_transform(X["Gender"])*

#*Encode 'Gegraohy' using OneHotEncoder. "Gegraohy" is in "2nd-column", hence [1]*

ct = **ColumnTransformer**(transformers = [("encoding", **OneHotEncoder**(), [1])], remainder = 'passthrough')

# *remainder = 'passthrough' for remaining columns to be unchanged*

X = **ct.fit\_transform**(X)

X = **np.array**(X) # *convert this output to NumPy array*

**print**(X)

X = X[:, 1:] # *Excluding 0-index column to avoid dummy-variable trap*

# *------------------ Data Split -----------------------*

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

X\_train, X\_test, y\_train, y\_test = **train\_test\_split**(X, y, test\_size= 0.2, random\_state = 0)

**11.2.3 XGBoost usin Python:** Apply the model

Here we are going to implement ***XGBoost*** by importing the *classifier* and creating an *object* of that classifier.

* Then we will ***fit*** this ***object*** to the ***training set*** and
* Then we will ***evaluate*** model ***performance*** making ***confusion matrix***.
* And we also apply ***K-fold-cross-validation*** for evaluating the ***model*** ***performance***
* Importing the Class: We import **XGBClassifier** from the module **xgboost** (we just installed it !!!).

**from** xgboost **import** XGBClassifier

* Parameters: When we're creating the object from **XGBClassifier**, we noticed that there is *very little information* about this class when we hit (ctrl+I or cmd+I), very *little documentation*. The only thing we get is the *list* of the ***parameters*** we can *input*.

|  |  |
| --- | --- |
| * Some parameters are: ***learning\_rate*** (as we had in deep learning), ***n\_estimators*** *(the number estimators, because* ***xgboost*** *is actually a Gradient Boosting Algorithm with Trees. So the number of estimators is actually the number of trees)* and then we have some other parameters but we're not going to play with those parameters now. |  |

* However, this is only an introduction to this very powerful algorithm but of course you can find a lot of information on the internet.
* Now we're not going to input any parameters we're going to be satisfied with the default values here in this **XGBClassifier** class.
* So we are going to take **max\_depth**= 3, **n\_estimators** =100 (one hundred trees) and the rest is fine because actually we have a binary outcome.
* Also you can do a little practice and try to do some Grid Search Parameter Tuning for example to find the optimal parameters for the learning\_rates or the n\_ estimators. It's exactly the same technique as how we did when we implemented Grid Search in the ***Previous*** ***Section***.
* We have another ***gamma*** parameter here that you can try to tune with grid search.
* Fitting this classified object to the training set: We just take our classifier object and fit the training-set. Exactly the same as when we implemented the ***other classification models***.

**claSifire\_XGB.fit**(X\_train, y\_train)

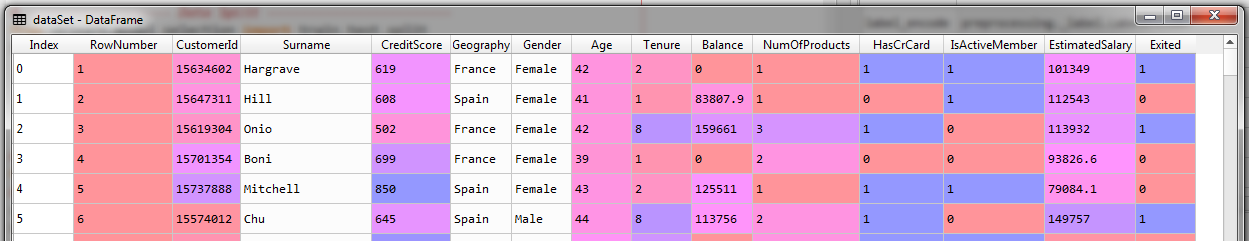
# *--------------- Fitting XGBoost in the training set ------------*

**from** xgboost **import** XGBClassifier

claSifire\_XGB = **XGBClassifier**()

**claSifire\_XGB.fit**(X\_train, y\_train)

* Prediction, confusion matrix, k-fold-cross-validation: All are copied from previous projects. Just change the classifier name.
* We make some *predictions* on the test-set
* We also calculate the *confusion-matrix*.
* Lastly we apply K-fold-cross-validation to get relevant performance metrics to assess the performance of our ***XGBoost*** model.



* The dataset contains some information of customers in a bank.
* The independent variables that we selected are all independent variables from ***credit score*** to ***estimated*** ***salary*** and the dependent variable is ***exited*** variable ***1*** for ***leave*** and ***0*** for ***stay***. These are the data of the previous six months that the Bank recorded.
* We were training this ***XGBoost*** model in the data set so that it understands the correlations between these information like the *credit score, the geography, the gender, age, the estimated salary* and the ***decision*** of the customer to ***leave*** the bank and therefore that's a classification problem.
* ***XGBoost*** will classify the customers between two classes the ones that leave and the ones that stay and then our goal is to make some *predictions* for the *future* *customers* and predict if they're going to *leave* the bank *yes* or *no*.

# *-------------------- Part 3 : Predictions and Evaluating the model ----------------------*

# *Predict*

y\_prd = **claSifire\_XGB.predict**(X\_test)

# *Making the confusion matrix use the function "confusion\_matrix"*

**from** sklearn.metrics **import** confusion\_matrix

cm = **confusion\_matrix**(y\_true= y\_test, y\_pred= y\_prd)

**print**(f"\nConfusion Matrix :\n {cm}")

# *parameters of cm: y\_true: Real values, y\_pred: Predicted value*

accURacy\_by\_Confusion\_matrix = (cm[0][0] + cm[1][1])/X\_test.shape[0]

**print**(f"\nAccuracy = {accURacy\_by\_Confusion\_matrix}%")

# *--------   K-folds cross-validation  -------------*

**from** sklearn.model\_selection **import** cross\_val\_score

accuRacies = **cross\_val\_score**(estimator=claSifire\_XGB, X = X\_train, y = y\_train, cv= 10)

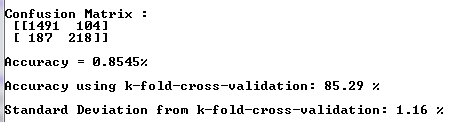
mean\_accu = **accuRacies.mean**()

std\_accu = **accuRacies.std**()

**print**("\nAccuracy using k-fold-cross-validation: {:.2f} %".**format**(mean\_accu\*100))

**print**("\nStandard Deviation from k-fold-cross-validation: {:.2f} %".**format**(std\_accu\*100))

* Result: We have a lot of correct predictions we have 1491 correct predictions of customers that don't leave the bank and 218 of customers that leave the bank.



* And then we have ***104 + 187*** incorrect predictions. The accuracy is ***(1709/2000)=0.855*** i.e. ***85.5%*** accuracy.
* We know that's not the most relevant accuracy. The relevant accuracy we get ***85.29%*** from K-fold-cross-validation.
* We also get ***1.16 % deviation*** i.e. our models accuracy range is ***84.13%*** to ***86.45%*** ,
* Besides it's very difficult to improve the accuracy because as we said before it is limited by the problem itself.

**All code at once (practiced version)**

# *----------------- XGBoost instead of ANN -----------------*

# *Library*

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

**import** matplotlib.pyplot **as** pLt

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

# *-------------------------- Part 1 : Data Preprocessing -----------------------------*

# *Data Extract*

dataSet = **pd.read\_csv**("Churn\_Modelling.csv")

# *X = dataSet.iloc[:, 3:-1].values # this can be used too*

X = dataSet.iloc[:, 3:13].values # *all columns from index 3, excluding 13 indexed column*

y = dataSet.iloc[:, 13].values # *the last column*

# *------------- Encode Categorical Data  -----------*

**from** sklearn.preprocessing **import** LabelEncoder, OneHotEncoder

**from** sklearn.compose **import** ColumnTransformer

#*Encode "Gender" using LabelEncoder."Gender" is in "3rd-column", hence X[:, 2]*

label\_encode = **LabelEncoder**()

# *Following is applicable to numpy array, if we used "X = dataSet.iloc[:, 3:13]"*

# *X = np.array(X) # it is needed if X is not an Arry. i.e. ".values" not applied*

X[:, 2] = **label\_encode.fit\_transform**(X[:, 2])

**print**(X)

# *For a data-set we can still encode it using Columns "key"*

# *X["Gender"] = label\_encode.fit\_transform(X["Gender"])*

#*Encode 'Gegraohy' using OneHotEncoder. "Gegraohy" is in "2nd-column", hence [1]*

ct = **ColumnTransformer**(transformers = [("encoding", **OneHotEncoder**(), [1])], remainder = 'passthrough')

# *remainder = 'passthrough' for remaining columns to be unchanged*

X = **ct.fit\_transform**(X)

X = **np.array**(X) # *convert this output to NumPy array*

**print**(X)

X = X[:, 1:] # *Excluding 0-index column to avoid dummy-variable trap*

# *------------------ Data Split -----------------------*

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

X\_train, X\_test, y\_train, y\_test = **train\_test\_split**(X, y, test\_size= 0.2, random\_state = 0)

# *---------------------------- Part 2 :Fitting XGBoost in the training set----------------------------*

**from** xgboost **import** XGBClassifier

claSifire\_XGB = **XGBClassifier**()

**claSifire\_XGB.fit**(X\_train, y\_train)

# *--------------------------- Part 3 : Predictions and Evaluating the model ---------------------------*

# *----- Predict -----*

y\_prd = **claSifire\_XGB.predict**(X\_test)

# *-------- confusion matrix ------*

**from** sklearn.metrics **import** confusion\_matrix

cm = **confusion\_matrix**(y\_true= y\_test, y\_pred= y\_prd)

**print**(f"\nConfusion Matrix :\n {cm}")

# *parameters of cm: y\_true: Real values, y\_pred: Predicted value*

accURacy\_by\_Confusion\_matrix = (cm[0][0] + cm[1][1])/X\_test.shape[0]

**print**(f"\nAccuracy = {accURacy\_by\_Confusion\_matrix}%")

# *--------   K-folds cross-validation  -------------*

**from** sklearn.model\_selection **import** cross\_val\_score

accuRacies = **cross\_val\_score**(estimator=claSifire\_XGB, X = X\_train, y = y\_train, cv= 10)

mean\_accu = **accuRacies.mean**()

std\_accu = **accuRacies.std**()

**print**("\nAccuracy using k-fold-cross-validation: {:.2f} %".**format**(mean\_accu\*100))

**print**("\nStandard Deviation from k-fold-cross-validation: {:.2f} %".**format**(std\_accu\*100))

# *python prctc\_XGBoost.py*