20MIA1031
SANJAY.M

Predictive Analytics with Case Studies(CSE3085)-F1

Material for Assignment-3

- **Malware**

Efficient Feature Extraction for Next-Generation Malware Detection

- **Churn paper**

Model optimization analysis of customer churn prediction using machine learning algorithms with focus on feature reductions

- **Churn prediction execution**

- **LIGHT GBM ppt**

➢ Malware

Efficient Feature Extraction for Next-Generation Malware Detection

"Efficient feature extraction for Next-Generation Malware Detection" by Joshua Saxe and michael E. Locasto, 2011.

This research paper that focuses on improving the efficiency and accuracy of malware detection by proposing a new approach for feature extraction.

Efficient feature extraction for Next-Generation malware Detection proposes a new feature extraction approach for malware detection using control flow graphs (CFGs).

The authors implemented their proposed approach in a tool called Bitshred, which extracts features from CFGs. Bitshred creates a hash of each node in the CFG and then performs pairwise comparisons between nodes in different CFGs to determine if they have similar control flow paths. Bitshred then aggregates similar nodes into shreds and uses the shreds as features to represent the malware samples.

To evaluate their approach, the authors used a large dataset of malware samples and compared the performance of Bitshred with existing feature extraction methods, such as n-gram and byte frequency histograms. The authors measures the detection accuracy and time efficiency of each method.

They have used several machine Learning models to classify malware samples based on the extracted features. They evaluated the performance of those models, including support vector machines (SVM), Random forest, Logistic regression. They compared the performance of these models with and without the Bitshred features, as well as with other feature extraction methods.

Overall, the authors used a combination of machine Learning model and feature extraction methods to evaluate the effectiveness of their proposed approach for malware detection. They compared the performance of their method with existing approaches to demonstrate the efficiency and effectiveness of their approach.

The authors introduce the concept of feature extraction for malware detection and the limitations of existing methods.

The authors propose a new feature extraction approach that uses a control flow graph (CFG) to capture the high-level behaviour of the malware.

Implementation :- They explained how Bitshred creates CFGs for each binary instruction in a program and then extracts features from those graphs.

They creates a hash of each node in the CFG, and then performs pairwise comparisons between nodes in different CFGs to determine if they have similar CFG Paths.

→ Then they aggregates similar nodes into shreds and uses the shreds as features to represent the malware samples.

Evaluation :- They evaluated BitShreds performance on a large dataset of malware samples and compare it to existing feature extraction methods. They show the BitShred is faster and more accurate than existing methods.

They evalute the performance of their proposed approach by comparing it with existing feature extraction methods, such as n-grams and byte frequency histograms. They use a large dataset of malware samples and measure the detection accuracy and time efficiency of each method. They also compare the performance of several machine learning models, including support vector machines (SVMs), Randomforests, Logistic Regression, with and without the BitShred features.

The authors discuss the strengths and limitations of their approach and propose directions for feature research. The authors present their results and discuss the strengths and limitations of their approach. They show that BitShred is faster and more accurate than existing methods and that the machine learning models using BitShred features outperform those using other features extraction methods.

The methodology of the paper involves proposing a new feature extraction approach, implementing it in a tool, evaluating its performance using a large dataset of malware samples, and comparing it with existing method. The authors also use several machine learning models to classify malware samples based on the extracted features and compare their performance.

Churn prediction

Model optimization analysis of customer churn prediction using machine learning algorithms with focus on feature reductions

Report

⇒ model optimization analysis of customer churn prediction using machine learning Algorithms with focus on feature Reductions.

→ The customer churn problem is a significant challenges for industries, especially in the telecommunication industry. A churner quits the service provided by operators and yields no profit to any further extend.

→ Customer retention is one of the primary goals of crm and effective churn prediction is essential for acheiving this goal.

→ Churn prediction methods are highly based on AI classification algorithms, but high dimensionality and imbalanced datasets can accurate churn prediction.

→ especially boosting methods, demonstrate excellent results in handling imbalanced datasets in churn prediction.

→ Light GBM is a powerful algorithm that has shown significant ability to improve models.

→ Bayesian optimization and genetic optimization are efficient and robust optimization algorithms for hyperparameter tuning.

→ solve high precision problems and can be employed to increases classification accuracy while requiring a low computing cost of the data mining process.

→ popular feature reduction methods include Autoencoders, xgboost.

→ we have steps(5) involed → Data preprocessing, EDA
→ feature reduction, tuning
→ model Prediction, development.

→ All ML algorithms like Adaboost, Svm, decision tree etc.,

→ we got Light GBM is best accurated.

→ Xiao et al → proposed a feature selection model of churn prediction using an ensemble algorithm with a dynamic environment.
→ imbalanced dataset.

→ saghir et al. → Churn prediction using a method of neural network.

→ Hoppner et al → presented models of churn detection for achieving a high-profit model based on the expected optimum profit for the churn of customers.

→ Datasets from various industries, such as finance, retail, gaming music, e-commerce has been used for churn prediction.

→ Data cleaning techiques and handling of categorical and imbalanced datasets are essential preprocessing methods in churn prediction.

→ imbalanced learning techniques, under-sampling, upsampling and algorithm approaches can be used to handle imbalanced datasets.

→ The paper approach for churn prediction in telecom industry.

→ That results shows that the outperforms the benchmark models in terms of accuracy, precision, recall.

→ This feature reduction steps is crucial in reducing the computational costs and storage space.

→ Approaches like PCA, Autoencodor, xgboost has been evaluated.

→ Bayesian optimization & genetic algorithm are more efficient and time effective compared to other models.

→ finally customer life time value analysis has been preformed to rank the churner and allocate resources efficiently.

→ The proposed frame work has been tested on three datasets IBM, Bank, orange, it given accuracy, f1 score.

→ In future work more advanced and sophisticated methods can be used.

→ different classifiers and improve the prediction performance.

→ The model can be applied to healthcare, finance, ecomerce prediction of customer churn and enhance customer retention strategies.

→ finally the proposed frame work can be used as a basis for developing a decision support system for customer churn management.

→ Ligh BGM is a sequential boosting of decision tree based ensemble machine learning algorithm.

→ The algorithm uses a gradient boosting approach.

→ The tree are built successively in order to minimize each previous tree loss function.

Genratic initial population of Chromosemes
↓
Calculate fitness
↓
Terminate ← Optimum Solution
↓
Select fitted chromosomes
↓
Genetic operation
↓
Next Generation

Genetic optimization

Bayesian optimization.

Bayesian optimization
↓
Define the search space
↓
generate serogate model
↓
sample new Hyper parameters
↓
evaluate the generative function
↓
update data & Hyperparameters
↓
Stopping criteria Reached
↓
end

Methodology

Churn prediction → understand business and customers features → Gather the data
↓
Preprocess data
↓
Building ML Model
↓
evaluate mode
↓
make conclusion and Strategies of retention

Churn prediction execution

PA_DA3_churn_prediction_20MIA1031.ipynb

File Edit View Insert Runtime Tools Help Last edited on April 8

+ Code + Text

20MIA1031

SANJAY.M

...

Predictive Analytics with Case Studies(CSE3085)

Using machine learning for churn prediction

.

Can you create a machine learning model that forecasts which customers will leave the business?

Estimating whether or not consumers leave a bank is the goal. The shutting of the client's bank account is the action that constitutes customer abandonment.

+ Code + Text

Exploratory Data Analysis

```
[ ] # Installation of required libraries
    import pandas as pd
    import numpy as np
    import seaborn as sns
    import matplotlib.pyplot as plt

    from sklearn.linear_model import LogisticRegression
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.svm import SVC
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.ensemble import RandomForestClassifier
    from xgboost import XGBClassifier
    from lightgbm import LGBMClassifier
    from sklearn.ensemble import GradientBoostingClassifier
    from sklearn.model_selection import train_test_split
    from sklearn import preprocessing
    from sklearn.metrics import accuracy_score
    from sklearn.model_selection import KFold
    from sklearn.model_selection import cross_val_score, GridSearchCV
```

PA_DA3_churn_prediction_20MIA1031.ipynb

File Edit View Insert Runtime Tools Help Last edited on April 8

+ Code + Text

```
[ ] import warnings
    warnings.filterwarnings("ignore", category=DeprecationWarning)
    warnings.filterwarnings("ignore", category=FutureWarning)
    warnings.filterwarnings("ignore", category=UserWarning)

    %config InlineBackend.figure_format = 'retina'

    # to display all columns and rows:
    pd.set_option('display.max_columns', None); pd.set_option('display.max_rows', None);
```

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive
```

```
[ ] # Reading the dataset
    df = pd.read_csv("/content/drive/MyDrive/Churn_Modelling.csv", index_col=0)
    df.columns = map(str.lower, df.columns)
```

```
# The first 5 observation units of the data set were accessed.
df.head()
```

	customerid	surname	creditscore	geography	gender	age	tenure	balance	numofproducts	hascrcard	isactivemember	estimatedsalary	exited
RowNumber													
1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.88	1
2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.63	0
5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

Information on the dataset:

It has 12 variables and 10,000 observations. Information about clients is contained in independent variables. Customer abandonment status is a dependent variable.

** Variables: **

The output is unaffected by the value of **RowNumber**, which is the record (row) number. This column is going away.

The random values in **CustomerId** have no bearing on a customer's decision to leave the bank. This column is going away.

```
[ ] df.shape
```

```
(10000, 13)
```

```
[ ] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 1 to 10000
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   customerid       10000 non-null  int64
 1   surname          10000 non-null  object
 2   creditscore      10000 non-null  int64
 3   geography        10000 non-null  object
 4   gender           10000 non-null  object
 5   age              10000 non-null  int64
 6   tenure           10000 non-null  int64
 7   balance          10000 non-null  float64
 8   numofproducts    10000 non-null  int64
 9   hascrcard        10000 non-null  int64
 10  isactivemember   10000 non-null  int64
 11  estimatedsalary  10000 non-null  float64
 12  exited           10000 non-null  int64
dtypes: float64(2), int64(8), object(3)
memory usage: 1.1+ MB
```

```
df.describe()
```

	creditscore	age	tenure	balance	numofproducts	hascrcard	isactivemember	estimatedsalary	exited	NewAGT	NewEstimatedSalary	geography_Germany	geography_Spain	gender_Male
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881	0.203700	33.909000	8340.853323	0.250900	0.247700	0.545700
std	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818	0.402769	10.907118	4792.541068	0.433553	0.431698	0.497932
min	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000	0.000000	8.000000	0.965000	0.000000	0.000000	0.000000
25%	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000	0.000000	27.000000	4250.175833	0.000000	0.000000	0.000000
50%	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000	0.000000	32.000000	8349.492917	0.000000	0.000000	1.000000
75%	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500	0.000000	39.000000	12449.020625	1.000000	0.000000	1.000000
max	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000	1.000000	91.000000	16666.040000	1.000000	1.000000	1.000000

```
[ ] # The dependent variable was used to calculate the age variable's average.

    df.groupby("exited").agg("mean")
```

exited	customerid	creditscore	age	tenure	balance	numofproducts	hascrcard	isactivemember	estimatedsalary
0	1.569117e+07	651.853196	37.408389	5.033279	72745.296779	1.544267	0.707146	0.554565	99738.391772
1	1.569005e+07	645.351497	44.837997	4.932744	91108.539337	1.475209	0.699067	0.360825	101465.677531

```
[ ] # The average age variable was evaluated in relation to the gender variable.

    df.groupby("gender").agg({"age": "mean"})
```

gender	age
Female	39.238389
Male	38.658237

```
[ ] # The average of the dependent variable was evaluated in relation to the gender variable.

    df.groupby("gender").agg({"exited": "mean"})
```

gender	exited
Female	0.250715
Male	0.164559

```
# The dependent variable's average was studied in relation to the geography variable.

df.groupby("geography").agg({"exited": "mean"})
```

geography	exited
France	0.161548
Germany	0.324432
Spain	0.166734

```
[ ] # The dependant variable provides access to those who abandoned us. (exited == 1)
    churn = df[df["exited"] == 1]
```

```
[ ] churn.head()
```

RowNumber	customerid	surname	creditscore	geography	gender	age	tenure	balance	numofproducts	hascrcard	isactivemember	estimatedsalary	exited
1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.88	1
3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
6	15574012	Chu	645	Spain	Male	44	8	113755.78	2	1	0	149756.71	1
8	15656148	Obinna	376	Germany	Female	29	4	115046.74	4	1	0	119346.88	1
17	15737452	Romeo	653	Germany	Male	58	1	132602.88	1	1	0	5097.67	1

```
[ ] churn.shape
    (2037, 13)
```

```
[ ] # Apart to gender, who departed the most?
    churn.groupby("gender").agg({"exited": "count"})
```

gender	exited
Female	1139
Male	898

```
# Which country has left us the most?
churn.groupby("geography").agg({"exited": "count"})
```

geography	exited
France	810
Germany	814
Spain	413

```
[ ] # Access to those who do not leave us on the dependent variable. (exited == 0)
    non_churn = df[df["exited"] == 0]
```

```
[ ] non_churn.head()
```

RowNumber	customerid	surname	creditscore	geography	gender	age	tenure	balance	numofproducts	hascrcard	isactivemember	estimatedsalary	exited
2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.63	0
5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0
7	15592531	Bartlett	822	France	Male	50	7	0.00	2	1	1	10062.80	0
9	15792365	He	501	France	Male	44	4	142051.07	2	0	1	74940.50	0

```
[ ] non_churn.shape
    (7963, 13)
```

```
[ ] # Which country does not leave the most?
    non_churn.groupby("geography").agg({"exited": "count"})
```

geography	exited
France	4204
Germany	1695
Spain	2064

```
[ ] # Unique observation units were examined.
    df.nunique()
```

```
customerid         10000
surname             2932
creditscore          460
geography              3
gender                 2
age                   70
tenure                11
balance             6382
numofproducts          4
hascrcard              2
isactivemember         2
estimatedsalary     9999
exited                 2
dtype: int64
```

```
[ ] # What are the most commonly used surnames?

    df.groupby('surname')['surname'].count().sort_values(ascending=False).head(10)
```

```
surname
Smith       32
Martin      29
Scott       29
Walker      28
Brown       26
Shih        25
Genovese    25
Yeh         25
Wright      24
Maclean     24
Name: surname, dtype: int64
```

```
# The most commonly used surname was examined and observed to be non-multiplexing.

df[df["surname"] == "Smith"]
```

RowNumber	customerid	surname	creditscore	geography	gender	age	tenure	balance	numofproducts	hascrcard	isactivemember	estimatedsalary	exited
235	15704769	Smith	585	France	Female	67	5	113978.97	2	0	1	93146.11	0
480	15797736	Smith	658	France	Male	29	4	80262.60	1	1	1	20612.82	0
746	15640059	Smith	606	France	Male	40	5	0.00	2	1	1	70899.27	0
1065	15699598	Smith	723	France	Female	20	4	0.00	2	1	1	140385.33	0
1757	15670702	Smith	618	France	Male	37	2	168178.21	2	0	1	101273.23	0
2134	15726354	Smith	688	France	Female	32	6	123157.95	1	1	0	172531.23	0

```
[ ] # They were grouped by gender variable and looked at how many years on average they were clients.

    df.groupby(["gender", "exited"]).agg({"tenure" : "mean"})
```

gender	exited	tenure
Female	0	4.977086
	1	4.933275
Male	0	5.075236
	1	4.932071

```
[ ] # Min, mean and max values of all variables were taken according to the dependent variable.

    df.groupby("exited").agg(["min","mean","max"])
```

	customerid			creditscore			age			tenure			balance			numofproducts			hascrcard			isactivemember			estimatedsalary		
	min	mean	max	min	mean	max	min	mean	max	min	mean	max	min	mean	max	min	mean	max	min	mean	max	min	mean	max	min	mean	max
exited																											
0	15565701	1.569117e+07	15815690	405	651.853196	850	18	37.408389	92	0	5.033279	10	0.0	72745.296779	221532.80	1	1.544267	3	0	0.707146	1	0	0.554565	1	90.07	99738.391772	199992.48
1	15565706	1.569005e+07	15815656	350	645.351497	850	18	44.837997	84	0	4.932744	10	0.0	91108.539337	250898.09	1	1.475209	4	0	0.699067	1	0	0.360825	1	11.58	101465.677531	199808.10

```
[ ] # Isactivemember is grouped according to the hascrcard variables and the dependent variable is examined.

    df.groupby(["isactivemember", "hascrcard"]).agg({"exited" : "count"})
```

isactivemember	hascrcard	exited
0	0	1401
	1	3448
1	0	1544
	1	3607

```
[ ] # Isactivemember is grouped according to hascrcard variables and the balance variable is examined.

    df.groupby(["isactivemember", "hascrcard"]).agg({"balance" : "mean"})
```

isactivemember	hascrcard	balance
0	0	77825.424525
	1	76853.588646
1	0	78007.318381
	1	74962.849983

```
[ ] # The balance variable was examined according to the gender variable.

    df.groupby("gender").agg({"balance": "mean"})
```

gender	balance
Female	75659.369139
Male	77173.974506

```
# The age variable was divided into 5 parts and the age range in which there is the most abandonment was examined.

df["NewAge"] = pd.qcut(df['age'], 5)
df.groupby("NewAge")["exited"].value_counts()
```

```
NewAge          exited
(17.999, 31.0]  0        2191
                1         181
(31.0, 35.0]    0        1615
                1         166
(35.0, 40.0]    0        1927
                1         339
(40.0, 46.0]    0        1211
                1         485
(46.0, 92.0]    0        1019
                1         866
Name: exited, dtype: int64
```

```
[ ] # The dependent variable was studied according to the gender variable and age range variable.

    df.groupby(["gender","NewAge" ])["exited"].value_counts()
```

```
gender  NewAge          exited
Female  (17.999, 31.0]  0         959
                        1         107
        (31.0, 35.0]    0         670
                        1          96
        (35.0, 40.0]    0         818
                        1         189
        (40.0, 46.0]    0         525
                        1         263
        (46.0, 92.0]    1         484
                        0         432
Male    (17.999, 31.0]  0        1232
                        1          74
        (31.0, 35.0]    0         945
                        1          70
        (35.0, 40.0]    0        1109
                        1         150
        (40.0, 46.0]    0         686
                        1         222
        (46.0, 92.0]    0         587
                        1         382
Name: exited, dtype: int64
```

```
[ ] # How many people whose balance is 0 and do not leave?

    df[(df["balance"] == 0) & (df["exited"] == 0)].shape
```

```
(3117, 14)
```

```
[ ] # How many people whose balance is 0 leave?

    df[(df["balance"] == 0) & (df["exited"] == 1)].shape
```

```
(500, 14)
```

```
[ ]
df.corr()
```

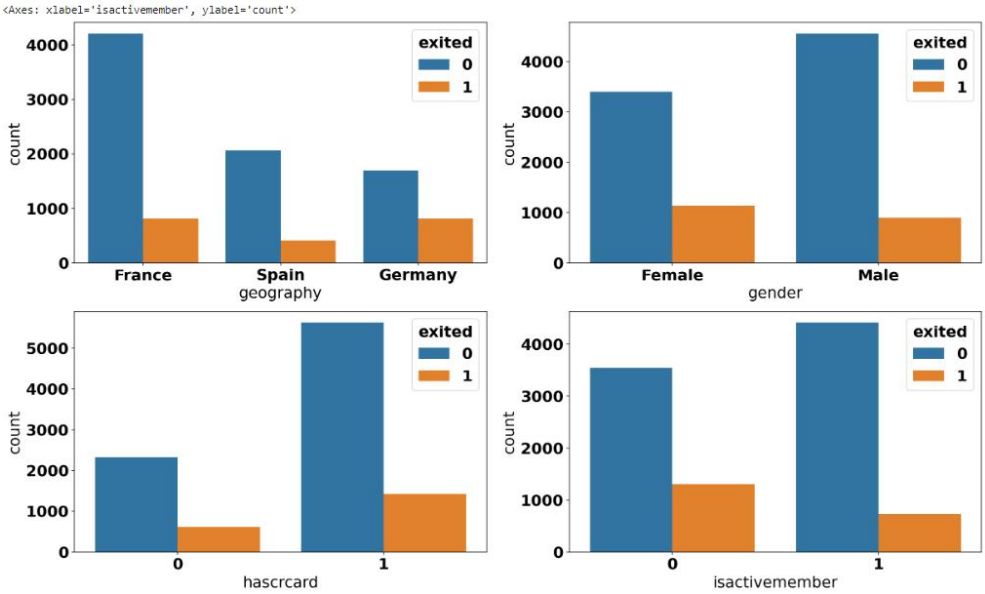
	customerid	creditscore	age	tenure	balance	numofproducts	hascrcard	isactivemember	estimatedsalary	exited
customerid	1.000000	0.005308	0.009497	-0.014883	-0.012419	0.016972	-0.014025	0.001665	0.015271	-0.006248
creditscore	0.005308	1.000000	-0.003965	0.000842	0.006268	0.012238	-0.005458	0.025651	-0.001384	-0.027094
age	0.009497	-0.003965	1.000000	-0.009997	0.028308	-0.030680	-0.011721	0.085472	-0.007201	0.285323
tenure	-0.014883	0.000842	-0.009997	1.000000	-0.012254	0.013444	0.022583	-0.028362	0.007784	-0.014001
balance	-0.012419	0.006268	0.028308	-0.012254	1.000000	-0.304180	-0.014858	-0.010084	0.012797	0.118533
numofproducts	0.016972	0.012238	-0.030680	0.013444	-0.304180	1.000000	0.003183	0.009612	0.014204	-0.047820
hascrcard	-0.014025	-0.005458	-0.011721	0.022583	-0.014858	0.003183	1.000000	-0.011866	-0.009933	-0.007138
isactivemember	0.001665	0.025651	0.085472	-0.028362	-0.010084	0.009612	-0.011866	1.000000	-0.011421	-0.156128
estimatedsalary	0.015271	-0.001384	-0.007201	0.007784	0.012797	0.014204	-0.009933	-0.011421	1.000000	0.012097
exited	-0.006248	-0.027094	0.285323	-0.014001	0.118533	-0.047820	-0.007138	-0.156128	0.012097	1.000000

Data Visualization

```
[ ]
```

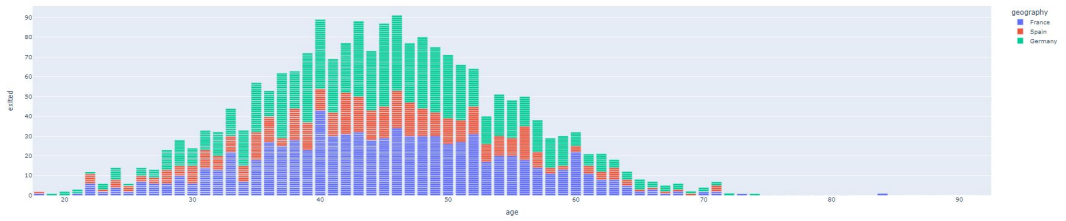
```
# Plotted the categorical variables on the basis of the graph of the column according to the dependent variable.
fig, axarr = plt.subplots(2, 2, figsize=(20, 12))
sns.countplot(x='geography', hue = 'exited',data = df, ax=axarr[0][0])
sns.countplot(x='gender', hue = 'exited',data = df, ax=axarr[0][1])
sns.countplot(x='hascrcard', hue = 'exited',data = df, ax=axarr[1][0])
sns.countplot(x='isactivemember', hue = 'exited',data = df, ax=axarr[1][1])
```

```
<Axes: xlabel='isactivemember', ylabel='count'>
```

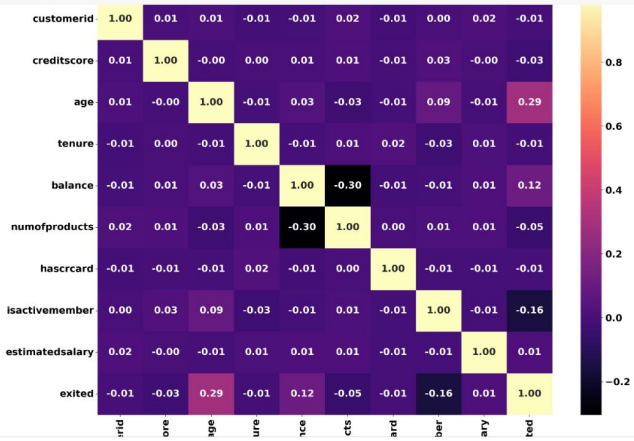



```
# The distribution of the dependent variable according to the gender variable is plotted in the pie chart.

import plotly.express as px
fig = px.pie(df, values =df.groupby("gender")["exited"].value_counts(), names = ["Female,0","Female 1","Male,0","Male,1"])
fig.show()
```

```
[ ] # Dependent variable was plotted according to age and geography variable.

    import plotly.express as px
    fig = px.bar(df,y = "exited", x = "age" , color = "geography")
    fig.show()
```

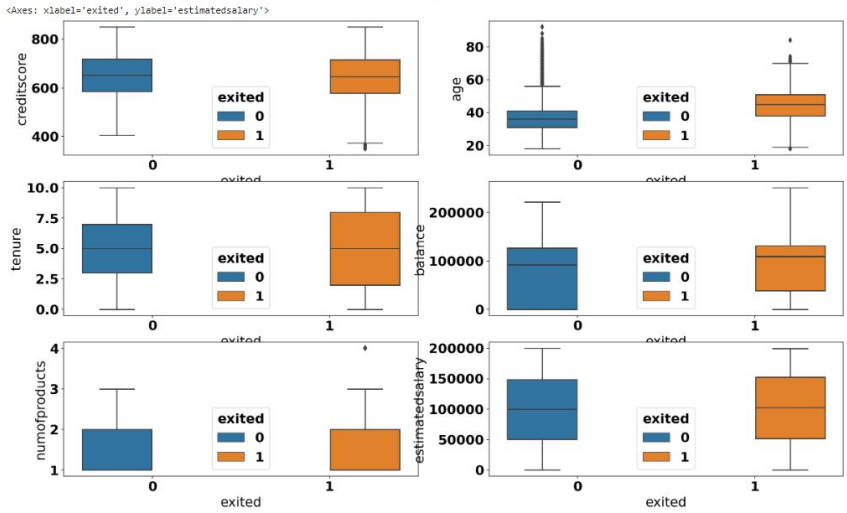



```
# Correlation Matrix

f, ax = plt.subplots(figsize= [20,15])
sns.heatmap(df.corr(), annot=True, fmt=".2f", ax=ax, cmap = "magma" )
ax.set_title("Correlation Matrix", fontsize=20)
plt.show()
```




```
# Boxplot graph for outlier observation analysis

fig, axarr = plt.subplots(3, 2, figsize=(20, 12))
sns.boxplot(y='creditscore',x = 'exited', hue = 'exited',data = df, ax=axarr[0][0])
sns.boxplot(y='age',x = 'exited', hue = 'exited',data = df , ax=axarr[0][1])
sns.boxplot(y='tenure',x = 'exited', hue = 'exited',data = df, ax=axarr[1][0])
sns.boxplot(y='balance',x = 'exited', hue = 'exited',data = df, ax=axarr[1][1])
sns.boxplot(y='numofproducts',x = 'exited', hue = 'exited',data = df, ax=axarr[2][0])
sns.boxplot(y='estimatedsalary',x = 'exited', hue = 'exited',data = df, ax=axarr[2][1])
```

```
<Axes: xlabel='exited', ylabel='estimatedsalary'>
```



Data Preprocessing

Missing and Outlier Observation Analysis

```
[ ] # Missing Observation Analysis
    df.isnull().sum()
```

```
customerid         0
surname            0
creditscore        0
geography          0
gender             0
age                0
tenure             0
balance            0
numofproducts      0
hascrcard          0
isactivemember     0
estimatedsalary    0
exited             0
NewAge             0
dtype: int64
```

```
[ ] # Outlier Observation Analysis
    for feature in df[['creditscore','tenure', 'balance','estimatedsalary']]:

        Q1 = df[feature].quantile(0.25)
        Q3 = df[feature].quantile(0.75)
        IQR = Q3-Q1
        lower = Q1- 1.5*IQR
        upper = Q3 + 1.5*IQR

        if df[(df[feature] > upper)].any(axis=None):
            print(feature,"yes")
        else:
            print(feature, "no")
```

```
creditscore no
tenure no
balance no
estimatedsalary no
```

Feature Engineering

```
[ ] df["NewAGT"] = df["age"] - df["tenure"]
    df["CreditsScore"] = pd.qcut(df['creditscore'], 10, labels = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
    df["AgeScore"] = pd.qcut(df['age'], 8, labels = [1, 2, 3, 4, 5, 6, 7, 8])
    df["BalanceScore"] = pd.qcut(df['balance'].rank(method="first"), 10, labels = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
    df["EstSalaryScore"] = pd.qcut(df['estimatedsalary'], 10, labels = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
    df["NewEstimatedSalary"] = df["estimatedsalary"] / 12
```

```
[ ] df.head()
```

RowNumber	customerid	surname	creditscore	geography	gender	age	tenure	balance	numofproducts	hascrcard	isactivemember	estimatedsalary	exited	NewAge	NewAGT	CreditsScore	AgeScore	BalanceScore	EstSalaryScore	NewEstimatedSalary
1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.88	1	(40.0, 46.0]	40	4	6	1	6	8445.740000
2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0	(40.0, 46.0]	40	4	6	5	6	9378.548333
3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.57	1	(40.0, 46.0]	34	1	6	10	6	9494.297500
4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.63	0	(35.0, 40.0]	38	7	5	1	5	7818.885833
5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0	(40.0, 46.0]	41	10	6	8	4	6590.341667

One Hot Encoding

```
[ ] df = pd.get_dummies(df, columns =["geography", "gender"], drop_first = True)
```

```
df.head()
```

RowNumber	customerid	surname	creditscore	age	tenure	balance	numofproducts	hascrcard	isactivemember	estimatedsalary	exited	NewAge	NewAGT	CreditsScore	AgeScore	BalanceScore	EstSalaryScore	NewEstimatedSalary	geography_Germany	geography_Spain	gender_Male
1	15634602	Hargrave	619	42	2	0.00	1	1	1	101348.88	1	(40.0, 46.0]	40	4	6	1	6	8445.740000	0	0	0
2	15647311	Hill	608	41	1	83807.86	1	0	1	112542.58	0	(40.0, 46.0]	40	4	6	5	6	9378.548333	0	1	0
3	15619304	Onio	502	42	8	159660.80	3	1	0	113931.57	1	(40.0, 46.0]	34	1	6	10	6	9494.297500	0	0	0
4	15701354	Boni	699	39	1	0.00	2	0	0	93826.63	0	(35.0, 40.0]	38	7	5	1	5	7818.885833	0	0	0
5	15737888	Mitchell	850	43	2	125510.82	1	1	1	79084.10	0	(40.0, 46.0]	41	10	6	8	4	6590.341667	0	1	0

```
[ ] df = df.drop(["customerid","surname","NewAge"], axis = 1)
```

```
[ ] df.head()
```

RowNumber	creditscore	age	tenure	balance	numofproducts	hascrcard	isactivemember	estimatedsalary	exited	NewAGT	CreditsScore	AgeScore	BalanceScore	EstSalaryScore	NewEstimatedSalary	geography_Germany	geography_Spain	gender_Male
1	619	42	2	0.00	1	1	1	101348.88	1	40	4	6	1	6	8445.740000	0	0	0
2	608	41	1	83807.86	1	0	1	112542.58	0	40	4	6	5	6	9378.548333	0	1	0
3	502	42	8	159660.80	3	1	0	113931.57	1	34	1	6	10	6	9494.297500	0	0	0
4	699	39	1	0.00	2	0	0	93826.63	0	38	7	5	1	5	7818.885833	0	0	0
5	850	43	2	125510.82	1	1	1	79084.10	0	41	10	6	8	4	6590.341667	0	1	0

Scaling

```
[ ] cat_df = df[["geography_Germany", "geography_Spain", "gender_Male", "hascrcard","isactivemember"]]
```

```
[ ] cat_df.head()
```

RowNumber	geography_Germany	geography_Spain	gender_Male	hascrcard	isactivemember
1	0	0	0	1	1
2	0	1	0	0	1
3	0	0	0	1	0
4	0	0	0	0	0
5	0	1	0	1	1

```
y = df["exited"]
X = df.drop(["exited","geography_Germany", "geography_Spain", "gender_Male", "hascrcard","isactivemember"], axis = 1)
cols = X.columns
index = X.index
```

```
[ ] X.head()
```

	creditscore	age	tenure	balance	numofproducts	estimatedsalary	NewAGT	CreditsScore	AgeScore	BalanceScore	EstSalaryScore	NewEstimatedSalary
RowNumber												
1	619	42	2	0.00	1	101348.88	40	4	6	1	6	8445.740000
2	608	41	1	83807.86	1	112542.58	40	4	6	5	6	9378.548333
3	502	42	8	159660.80	3	113931.57	34	1	6	10	6	9494.297500
4	699	39	1	0.00	2	93826.63	38	7	5	1	5	7818.885833
5	850	43	2	125510.82	1	79084.10	41	10	6	8	4	6590.341667

```
[ ] from sklearn.preprocessing import RobustScaler
    transformer = RobustScaler().fit(X)
    X = transformer.transform(X)
    X = pd.Dataframe(X, columns = cols, index = index)
```

```
[ ] X = pd.concat([X,cat_df], axis = 1)
```

```
[ ] X.head()
```

	creditscore	age	tenure	balance	numofproducts	estimatedsalary	NewAGT	CreditsScore	AgeScore	BalanceScore	EstSalaryScore	NewEstimatedSalary	geography_Germany	geography_Spain	gender_Male	hascrcard	isactivemember
RowNumber																	
1	-0.246269	0.416667	-0.75	-0.761480	0.0	0.011739	0.666667	-0.2	0.50	-0.9	0.1	0.011739	0	0	0	1	1
2	-0.328358	0.333333	-1.00	-0.104906	0.0	0.125512	0.666667	-0.2	0.50	-0.1	0.1	0.125512	0	1	0	0	1
3	-1.119403	0.416667	0.75	0.489346	2.0	0.139630	0.166667	-0.8	0.50	0.9	0.1	0.139630	0	0	0	1	0
4	0.350746	0.166667	-1.00	-0.761480	1.0	-0.064717	0.500000	0.4	0.25	-0.9	-0.1	-0.064717	0	0	0	0	0
5	1.477612	0.500000	-0.75	0.221806	0.0	-0.214561	0.750000	1.0	0.50	0.5	-0.3	-0.214561	0	1	0	1	1

```
y.head()
```

```
RowNumber
1    1
2    0
3    1
4    0
5    0
Name: exited, dtype: int64
```

```
[ ] print(X.shape, y.shape)
```

```
(10000, 17) (10000,)
```

Train-Test Separation & Balancing

```
[ ] # Train-Test Separation
    X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                        test_size=0.20,
                                                        random_state=12345)
```

```
[ ] print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(8000, 17) (2000, 17) (8000,) (2000,)
```

Modelling

```
[ ]
```

```
models = []
models.append(('LR', LogisticRegression(random_state = 12345)))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier(random_state = 12345)))
models.append(('RF', RandomForestClassifier(random_state = 12345)))
models.append(('SVM', SVC(gamma='auto', random_state = 12345)))
models.append(('XGB', GradientBoostingClassifier(random_state = 12345)))
models.append(("LightGBM", LGBMClassifier(random_state = 12345)))

# evaluate each model in turn
results = []
names = []
```

Accuracy Score

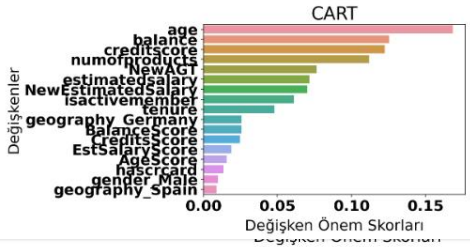
```
[ ] for name, model in models:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        msg = "%s: (%f)" % (name, accuracy)
        print(msg)
```

```
[ ] LR: (0.812000)
    KNN: (0.830000)
    CART: (0.788500)
    RF: (0.849000)
    SVM: (0.843500)
    XGB: (0.856500)
    LightGBM: (0.856000)
```

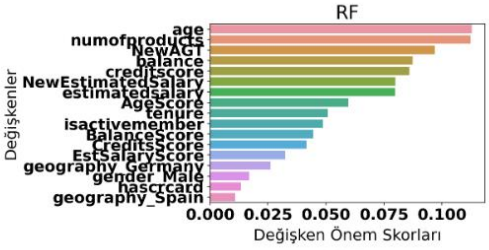
Variable Importance Levels of All Models

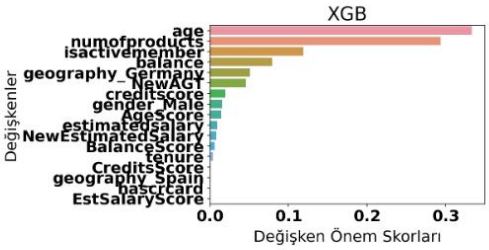
```
[ ] models2 = []
    models2.append(('CART', DecisionTreeClassifier( random_state = 12345)))
    models2.append(('RF', RandomForestClassifier( random_state = 12345)))
    models2.append(('XGB', GradientBoostingClassifier( random_state = 12345)))
    models2.append(("LightGBM", LGBMClassifier( random_state = 12345)))
```

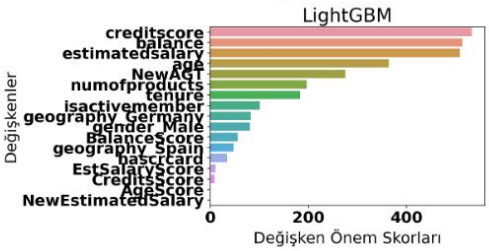
```
for name, model in models2:
        base = model.fit(X_train,y_train)
        y_pred = base.predict(X_test)
        acc_score = accuracy_score(y_test, y_pred)
        feature_imp = pd.Series(base.feature_importances_,
                        index=X.columns).sort_values(ascending=False)

        sns.barplot(x=feature_imp, y=feature_imp.index)
        plt.xlabel('Değişken Önem Skorları')
        plt.ylabel('Değişkenler')
        plt.title(name)
        plt.show()
```



[ ]







Model Tuning

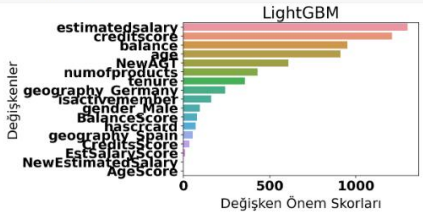
```
[ ] # Hyperparameters have previously been obtained with the help of GridSearchCV.
    models = []
    models.append(('XGB', GradientBoostingClassifier(random_state = 12345,learning_rate = 0.05, max_depth = 5, min_samples_split = 2, n_estimators = 500, subsample = 0.8)))
    models.append(("LightGBM", LGBMClassifier(random_state = 12345,  learning_rate = 0.05, max_depth = 3, n_estimators = 1000)))

    # evaluate each model in turn
    results = []
    names = []

    for name, model in models:
            model.fit(X_train, y_train)
            y_pred = model.predict(X_test)
            accuracy = accuracy_score(y_test, y_pred)
            msg = "%s: (%f)" % (name, accuracy)
            print(msg)
```

```
XGB: (0.858500)
LightGBM: (0.859000)
```

Model Tuning Variable Importance Level

```
models2 = []
models2.append(("LightGBM", LGBMClassifier(random_state = 12345,  learning_rate = 0.05, max_depth = 3, n_estimators = 1000)))

# evaluate each model in turn
results = []
names = []

for name, model in models2:
        base = model.fit(X_train,y_train)
        y_pred = base.predict(X_test)
        acc_score = accuracy_score(y_test, y_pred)
        feature_imp = pd.Series(base.feature_importances_,
                        index=X.columns).sort_values(ascending=False)

        sns.barplot(x=feature_imp, y=feature_imp.index)
        plt.xlabel('Değişken Önem Skorları')
        plt.ylabel('Değişkenler')
        plt.title(name)
        plt.show()
```

```
[ ]
```



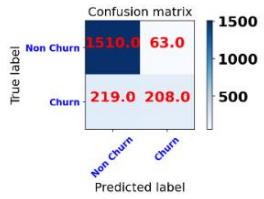
Confusion Matrix

```
[ ] from sklearn.metrics import  accuracy_score, f1_score, precision_score,confusion_matrix, recall_score, roc_auc_score
    y_pred = model.predict(X_test)
    cm_xgb = confusion_matrix(y_test, y_pred=y_pred)

    TP = cm_xgb[1, 1]
    TN = cm_xgb[0, 0]
    FP = cm_xgb[0, 1]
    FN = cm_xgb[1, 0]
```

```
from matplotlib import rc,rcParams
def plot_confusion_matrix(cm, classes,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.rcParams.update({'font.size': 19})
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title,fontdict={'size':'16'})
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45,fontsize=12,color="blue")
    plt.yticks(tick_marks, classes,fontsize=12,color="blue")
    rc('font', weight='bold')
    fmt = '.1f'
    thresh = cm.max()
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="red")
```

```
[ ]
    plt.ylabel('True label',fontdict={'size':'16'})
    plt.xlabel('Predicted label',fontdict={'size':'16'})
    plt.tight_layout()
```

```
[ ] import itertools
    plot_confusion_matrix(confusion_matrix(y_test, y_pred=y_pred), classes=['Non Churn','Churn'],
                          title='Confusion matrix')
```




```
[ ] tn, fp, fn, tp = cm_xgb.ravel()
    print("True Negatives: ",tn)
    print("False Positives: ",fp)
    print("False Negatives: ",fn)
    print("True Positives: ",tp)
```

```
True Negatives:  1510
False Positives:  63
False Negatives:  219
True Positives:  208
```

```
df_pred = pd.DataFrame(data=[],columns=["y_test"])
df_pred["y_pred"] = y_pred
df_pred["y_test"] = y_test
df_pred.index = df_pred.index + 1
df_pred.head()
```

	y_test	y_pred
1	NaN	1
2	NaN	0
3	NaN	0
4	1.0	0
5	NaN	0

```
[ ] df_pred.loc[TN_predicts_indexes,"prediction_result"] = "TN"
    df_pred.loc[TP_predicts_indexes,"prediction_result"] = "TP"
    df_pred.loc[FP_predicts_indexes,"prediction_result"] = "FP"
    df_pred.loc[FN_predict_indexes,"prediction_result"] = "FN"
    df_pred.head()
```

	y_test	y_pred	prediction_result
1	NaN	1	NaN
2	NaN	0	NaN
3	NaN	0	NaN
4	1.0	0	FN
5	NaN	0	NaN

```
[ ] df_pred[df_pred["prediction_result"] == "FP"].head()
```

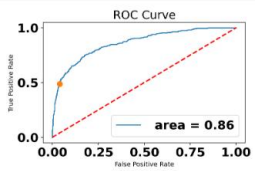
	y_test	y_pred	prediction_result
69	0.0	1	FP
156	0.0	1	FP
161	0.0	1	FP
271	0.0	1	FP
356	0.0	1	FP

```
[ ] df_pred[df_pred["prediction_result"] == "FN"].head()
```

	y_test	y_pred	prediction_result
4	1.0	0	FN
141	1.0	0	FN
145	1.0	0	FN
157	1.0	0	FN
168	1.0	0	FN

ROC Curve

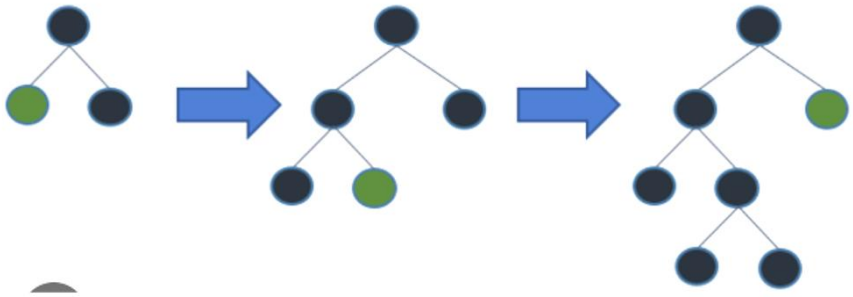
```
lbgm_tuned = model
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score, precision_recall_curve, roc_curve, auc, average_precision_score
y_pred_prob = lbgm_tuned.predict_proba(X_test)[:,1]
fig, ax = plt.subplots()
fpr, tpr, thresholds_roc = roc_curve(y_test, y_pred_prob)
roc_auc = auc(fpr,tpr)
ax.plot(fpr,tpr, label = " area = {:0.2f}".format(roc_auc))
ax.plot([0,1], [0,1], 'r', linestyle = "--", lw = 2)
ax.set_xlabel("False Positive Rate", fontsize = 10)
ax.set_ylabel("True Positive Rate", fontsize = 10)
ax.set_title("ROC Curve", fontsize = 18)
ax.legend(loc = 'best')

close_default = np.argmin(np.abs(thresholds_roc - 0.5))
ax.plot(fpr[close_default], tpr[close_default], 'o', markersize = 8)
plt.tight_layout()
```



+ Code + Text

LIGHT GBM PPT

LIGHTGBM



20MIA1031
Sanjay.M

INTRODUCTION

- LightGBM is a gradient boosting framework that uses tree based learning algorithms.
- It is designed to be distributed and efficient with the following advantages:
  - Faster training speed and higher efficiency.
  - Lower memory usage.
  - Better accuracy.
  - Support of parallel, distributed, and GPU learning.
  - Capable of handling large-scale data.
- LightGBM is a Supervised ensemble Machine Learning algorithm.

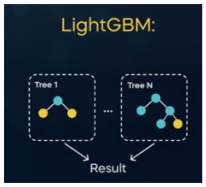
Why is LightGBM popular?

- It has become difficult for the traditional algorithms to give results fast, as the size of the data is increasing rapidly day by day.
- LightGBM is called "Light" because of its computation power and giving results faster.
- LightGBM is known to be both more efficient and scalable.
- It performs very well on small datasets as well as on large ones.

What is LightGBM used for?

- LightGBM is a gradient boosting ensemble method that is used by the Train Using AutoML tool and is based on decision trees.
- As with other decision tree-based methods, LightGBM can be used for both classification and regression.
- LightGBM is optimized for high performance with distributed systems.

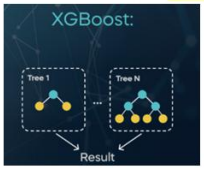
How is LightGBM different from XGBoost?



- LightGBM uses a novel technique of Gradient-based One-Side Sampling (GOSS) to filter out the data instances for finding a split value.
- while XGBoost uses pre-sorted algorithm & Histogram-based algorithm for computing the best split.

Is lgbm faster than XGBoost?

- There has been only a slight increase in accuracy and auc score by applying Light GBM over XGBOOST but there is a significant difference in the execution time for the training procedure.
- Light GBM is almost 7 times faster than XGBOOST and is a much better approach when dealing with large datasets.



- A properly-tuned LightGBM will most likely win in terms of performance and speed compared with random forest
- High value of estimators for random forest will affect it robustness, where as for GBM model will improve the model fit with your training data

Can LightGBM overfit?

- LightGBM uses the leaf-wise tree growth algorithm, while many other popular tools use depth-wise tree growth.
- Compared with depth-wise growth, the leaf-wise algorithm can converge much faster.
- However, the leaf-wise growth may be over-fitting if not used with the appropriate parameters.

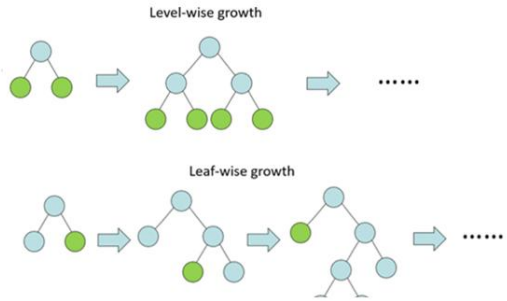
How does LightGBM treat null values?

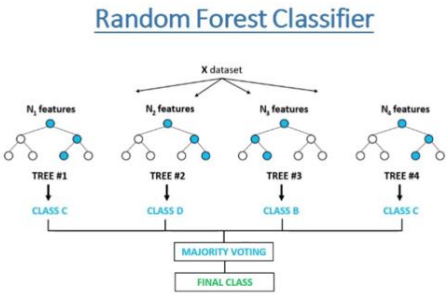
In lightgbm for categorical variables, "all negative values will be treated as missing values".
So the reason for often using values like -999 for null/na values is because of convention + usage of tree based algorithms like xgb/lgb.

What makes LightGBM lightning fast?

- There are three reasons why LightGBM is fast: Histogram based splitting, Gradient-based One-Side Sampling (GOSS), Exclusive Feature Bundling (EFB)

- Light GBM use histogram based **algorithm** i.e it buckets continuous feature values into discrete bins which fasten the training procedure.
- Lower memory usage: Replaces continuous values to discrete bins which result in lower memory usage.
- XGBoost was based on a level-wise growth algorithm





Github link my files

https://github.com/Sanjay3369/Predictive/tree/DA

Video link

https://drive.google.com/drive/folders/1q9X_G0W6wTKRSP7qlbW-I46XhU38caSi?usp=sharing