**2. Hyperparameter tuning and Feature engineering**

***1. Hyperparameter Tuning:***

Hyperparameters are parameters that are not learned by the model during training but are set prior to training. Optimizing these hyperparameters can significantly impact a model's performance. Here's how you can approach hyperparameter tuning.

*Grid Search:*

In grid search, you define a range of hyperparameter values, and the algorithm evaluates the model's performance for all possible combinations of these values. This can be computationally expensive but is thorough.

*Random Search:*

   Random search selects hyperparameter values randomly from predefined ranges. It's less computationally intensive than grid search and can often find good hyperparameters faster.

*Bayesian Optimization:*

   Bayesian optimization is a more sophisticated approach that uses probabilistic models to find the best set of hyperparameters. It's efficient and can often outperform grid or random search.

*Automated Hyperparameter Tuning:*

   You can also use automated machine learning libraries such as TPOT, Auto-Google to automate the hyperparameter tuning process.

*# Data manipulation*

import pandas as pd

import numpy as np

*# Modeling*

import lightgbm as lgb

*# Splitting data*

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

N\_FOLDS = 5

MAX\_EVALS = 5

features = pd.read\_csv('../input/home-credit-default-risk/application\_train.csv')

*# Sample 16000 rows (10000 for training, 6000 for testing)*

features = features.sample(n = 16000, random\_state = 42)

*# Only numeric features*

features = features.select\_dtypes('number')

*# Extract the labels*

labels = np.array(features['TARGET'].astype(np.int32)).reshape((-1, ))

features = features.drop(columns = ['TARGET', 'SK\_ID\_CURR'])

*# Split into training and testing data*

train\_features, test\_features, train\_labels, test\_labels = train\_test\_split(features, labels, test\_size = 6000, random\_state = 50)

print("Training features shape: ", train\_features.shape)

print("Testing features shape: ", test\_features.shape)

Training features shape: (10000, 104)

Testing features shape: (6000, 104)

*Create a training and testing dataset*

train\_set = lgb.Dataset(data = train\_features, label = train\_labels)

test\_set = lgb.Dataset(data = test\_features, label = test\_labels)

Training features shape: (10000, 104)

Testing features shape: (6000, 104)

*# Get default hyperparameters*

model = lgb.LGBMClassifier()

default\_params = model.get\_params()

*# Remove the number of estimators because we set this to 10000 in the cv call*

del default\_params['n\_estimators']

*# Cross validation with early stopping*

cv\_results = lgb.cv(default\_params, train\_set, num\_boost\_round = 10000, early\_stopping\_rounds = 100,

metrics = 'auc', nfold = N\_FOLDS, seed = 42)

import random

random.seed(50)

*# Randomly sample a boosting type*

boosting\_type = random.sample(param\_grid['boosting\_type'], 1)[0]

*# Set subsample depending on boosting type*

subsample = 1.0 if boosting\_type == 'goss' else random.sample(param\_grid['subsample'], 1)[0]

print('Boosting type: ', boosting\_type)

print('Subsample ratio: ', subsample)

Boosting type: goss

Subsample ratio: 1.0

Boosting type: goss

Subsample ratio: 1.0

import matplotlib.pyplot as plt

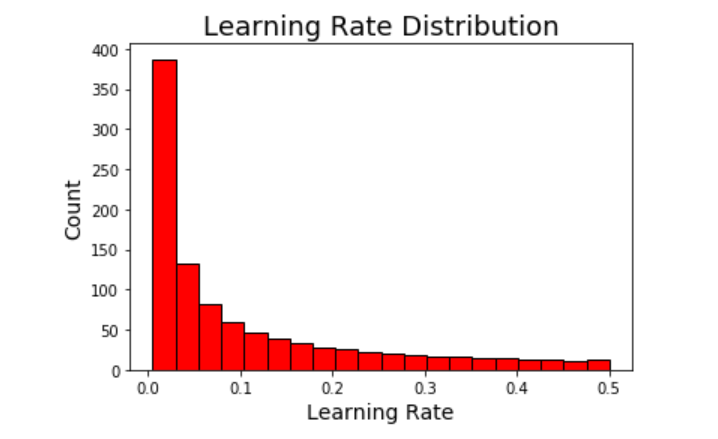
import seaborn as sns

%matplotlib inline

*# Learning rate histogram*

plt.hist(param\_grid['learning\_rate'], bins = 20, color = 'r', edgecolor = 'k');

plt.xlabel('Learning Rate', size = 14); plt.ylabel('Count', size = 14); plt.title('Learning Rate Distribution', size = 18);



a = 0

b = 0

*# Check number of values in each category*

for x **in** param\_grid['learning\_rate']:

*# Check values*

if x >= 0.005 **and** x < 0.05:

a += 1

elif x >= 0.05 **and** x < 0.5:

b += 1

print('There are **{}** values between 0.005 and 0.05'.format(a))

print('There are **{}** values between 0.05 and 0.5'.format(b))

There are 499 values between 0.005 and 0.05

There are 499 values between 0.05 and 0.5

a = 0

b = 0

*# Check number of values in each category*

for x **in** param\_grid['learning\_rate']:

*# Check values*

if x >= 0.005 **and** x < 0.05:

a += 1

elif x >= 0.05 **and** x < 0.5:

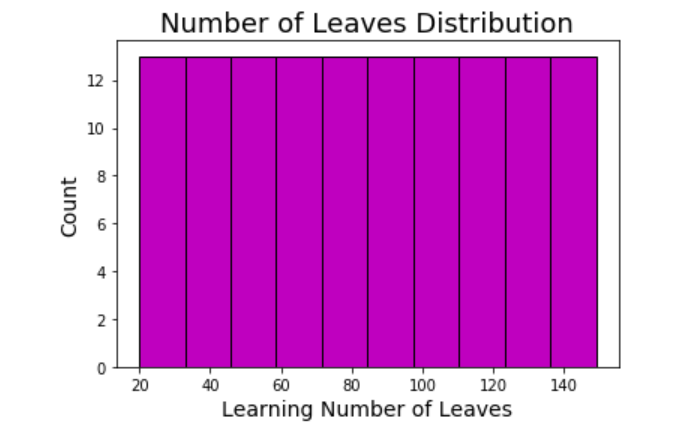
b += 1

print('There are **{}** values between 0.005 and 0.05'.format(a))

print('There are **{}** values between 0.05 and 0.5'.format(b))

There are 499 values between 0.005 and 0.05

There are 499 values between 0.05 and 0.5



***2. Feature Engineering:***

Feature engineering involves creating new features from the existing dataset or transforming existing features to provide more relevant information to

the model. Here are some techniques for feature engineering:

     Feature engineering involves creating new features from the existing  dataset or transforming existing features to provide more relevant information to the  model. Here are some techniques for feature engineering:

*Feature Selection:*

   Identify and keep only the most relevant features, removing any irrelevant or redundant ones. This can help reduce dimensionality and improve model performance.

*Feature Scaling:*

   Ensure that all features are on a similar scale, especially when using models sensitive to feature scales like gradient boosting or support vector machines

*Feature Extraction:*

  Techniques like Principal Component Analysis (PCA) or t-Distributed Stochastic Neighbor Embedding (t-SNE) can be used to extract relevant information from high-dimensional data.

*One-Hot Encoding:*

  Convert categorical variables into binary vectors (one-hot encoding) so that machine learning algorithms can work with them effectively.

*Feature Creation:*

   Sometimes, creating new features based on domain knowledge can be beneficial. For example: deriving a "customer loyalty score" from transaction history data.

*Handling Missing Data:*

 Decide on a strategy to handle missing data, which could involve imputation techniques like mean imputation, median imputation, or more advanced methods like K-nearest neighbor imputation.

Remember to validate the performance of your model using techniques like cross-validation to ensure that the improvements from hyperparameter tuning and feature engineering are statistically significant and generalize well to unseen data. It's also essential to iterate and experiment with different approaches to find the best combination of hyperparameters and feature engineering techniques for your specific problem.

import pandas as pd

import seaborn as sns

import numpy as np

from matplotlib import pyplot as plt

from matplotlib.ticker import MaxNLocator

from scipy import interp

import math

from scipy.stats import norm

from scipy import stats

import warnings

warnings.filterwarnings('ignore') *# Disabling warnings for clearer outputs.*

pd.options.display.max\_columns = 50 *# Pandas option to increase max number of columns to display.*

plt.style.use('ggplot') *# Setting default plot style.*

*# Read train and test data from csv files for visualization:*

v\_train = pd.read\_csv('/kaggle/input/titanic/train.csv')

v\_test = pd.read\_csv('/kaggle/input/titanic/test.csv')

idx = len(v\_train)

In [3]:

linkcode

*# Checking train and test sets*

display(v\_train.sample(3))

display(v\_test.sample(3))

| PassengerId | Survived | | Pclass | | Name | | Sex | | Age | | SibSp | | Parch | | Ticket | Fare | | Cabin | | Embarked | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 743 | 744 | | 0 | | 3 | | McNamee, Mr. Neal | | male | | 24.0 | | 1 | | 0 | 376566 | | 16.1000 | | NaN | | S |
| 547 | 548 | | 1 | | 2 | | Padro y Manent, Mr. Julian | | male | | NaN | | 0 | | 0 | SC/PARIS 2146 | | 13.8625 | | NaN | | C |
| 820 | 821 | | 1 | | 1 | | Hays, Mrs. | | female | | 52.0 | | 1 | | 1 | 12749 | | 93.5000 | | B69 | | S |
| PassengerId | | Pclass | | Name | | Sex | | Age | | SibSp | | Parch | | Ticket | | Fare | Cabin | | Embarked | |
| 143 | | 1035 | | 2 | | Beauchamp, Mr. Henry James | | male | | 28.0 | | 0 | | 0 | | 244358 | 26.00 | | NaN | | S | |
| 57 | | 949 | | 3 | | Abelseth, Mr. Olaus Jorgensen | | male | | 25.0 | | 0 | | 0 | | 348122 | 7.65 | | F G63 | | S | |
| 34 | | 926 | | 1 | | Mock, Mr. Philipp Edmund | | male | | 30.0 | | 1 | | 0 | | 13236 | 57.75 | | C78 | | C | |

*# Merging visualization datasets.*

v\_train.drop('PassengerId', axis=1, inplace=True)

v\_test.drop('PassengerId', axis=1, inplace=True)

v\_merged = pd.concat([v\_train, v\_test], sort=False).reset\_index(drop=True)

In [5]:

*# Checking merged shape:*

display(v\_merged.shape)

(1309, 11)

In [6]:

*# Checking features and target columns:*

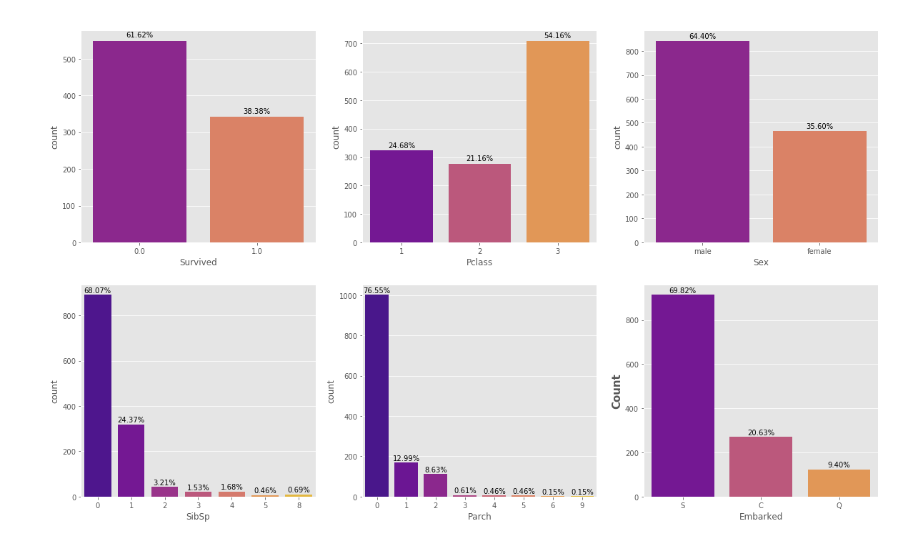
display(v\_merged.columns)

Index(['Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket',

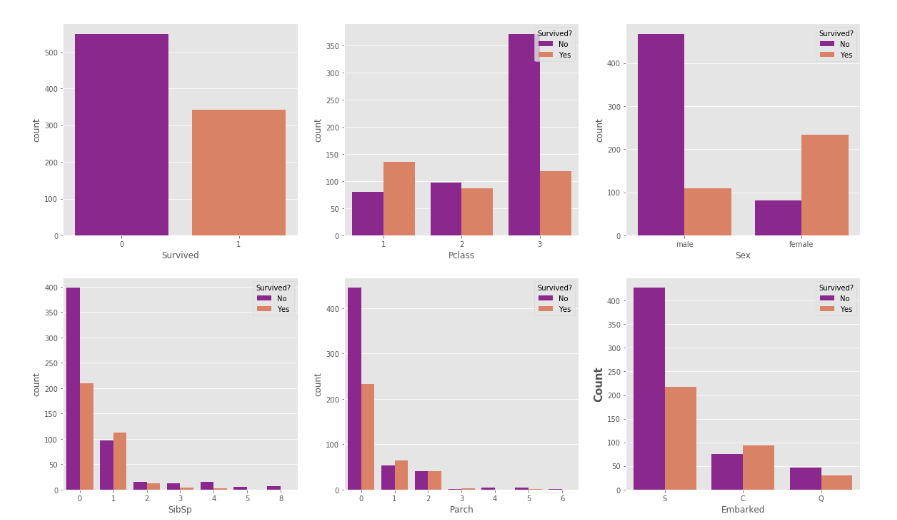
'Fare', 'Cabin', 'Embarked'],

dtype='object')

plotFrequency(cats)



plotsurvival(cats, v\_train)



plot\_3chart(v\_train, 'Age')

plot\_3chart(v\_train, 'Fare')

