Introduction: Business Problem

When a customer orders a product which is not available in the store or out-of-stock temporarily or due to lack of supply, the customer decides to wait until the desired product is available and there is a guaranteed delivery, then this scenario is called Backorder of the specific product. If the backorders are not handled promptly , there is a high chance of losing a customer to its competitor , along with impact on revenue,share market price etc. On the other hand, taking actions to satisfy or reduce backorders puts tremendous amounts of pressure on the different stages of supply chain management.Taking steps to satisfy or reduce backorders will lead to increased labor/production/transport/Warehouse costs etc.Machine Learning can identify patterns related to backorders before the customer orders. With this, the production can adjust to minimise delays for customer service and provide accurate dates to keep the customers informed. This predictive analysis approach gives the company ample time to react and enables them to satisfy the demands of customers and smooth the supply chain process.The case study “Backorder Prediction” deals with predicting the backorders of products by applying Machine Learning techniques to overcome or reduce the cost of backorders. We will identify parts with the highest chances of shortage so that we could present a high opportunity to improve the company's overall performance.This case study deals with investigating Machine Learning classifiers for imbalanced dataset where the chance of an item going into backorder is very rare when compared to items that do not.

Why Machine Learning is needed?

Through Machine Learning classification techniques, supply chain parameters could be analysed for different product lines.This allows Manufacturer/Company to predict and identify the products that would be unavailable in near future and then take the right step to neutralise the shortage.The predictions based on the ML Models improve with time as they get more data to work on.The degree of performance metrics of these predictive models can be improved by using ensembling techniques which uses diverse models combines them to produce improved results.

Problem Statement

Classify the product whether they would go into backorder(Yes or No) based on data like forecast sales,sales quantity ,threshold values,past performance etc.

ML Formulation

The task at hand is classifying whether a product will go to backorder or not for a given input data.This is a Binary Classification Problem hence consists of two target values :

- Yes: Represents that product will go to backorder.
- No: Represents that product will not go to backorder

Business Constraints

- No strict latency constraints.
- Misclassification may result in less effective supply chain management systems such as inaccurate demand forecasting and misclassification of backorder products.

Dataset Analysis

Dataset consists of 23 columns namely : SKU : stands for Stock Keeping Unit. It is a unique ID for each row

National_inv : Present Inventory level of the product

Lead time : Transit time of the product which means how long it takes for a shipment to be delivered at its final destination after it has been picked from the start point.

In_transit_qty : it is calculated based on the last picking slip or based on cumulative quantity.In simple words it is the amoun t of product in transit.

Forecast_3_months,Forecast_6_months,Forecast_9_months : Forecast of sales of the product for next 3,6,9 months respectively.

Sales_1_month,Sales_3_month,Sales_6_month : Sales of product in last 1,3,6,9 months respectively.

Min_bank : Minimum amount of stock recommended.

potential_issue : Problem/issue identified in the product/part

Pieces_past_due : Amount of parts of the product overdue if any.

Perf_6_months_avg,perf_12_months_avg:Product performance over past 6 months and 12 months respectively.

local_bo_qty : Amount of stock orders overdue

deck_risk,oe_constraint,ppap_risk,stop_auto_buy,rev_stop : Yes or No flags set for the products

went_to_backorder: Target Variable

Performance Metrics

In the case of Backorders,it happens very rarely that the products goes to backorder.So in this type of problems , Accuracy is not a useful metric since it works well with Balanced dataset. Since the dataset is imbalanced, the ideal performance metric would be Precision,Recall and F1 score , especially not to miss failure cases. F1 Score might be a better measure to use if we need to seek balance between Precision and Recall and also given that there is uneven class distribution(large number of Actual Negatives).

```
F1_score = 2 * ((Precision*Recall)/(Precision+Recall))
```

Precision : This can be thought of as , Out of postively predicted values, how many are actually positive.It is calculated as

```
Precision = True Positive / True Postive + False Positive
```

Recall : Recall can be thought of as, out of actual positives how many were detected by our model.It is calculated as

```
Recall = True Positive / True Positve + False Negative
```

Terms used in Precision and Recall are :

- True Positive : Product detected as Backorder and it turns out to be True.Benefits by predicting correctly the Backorders as Profit generated from such items is benefit.
- False Positive : Product detected as Backorder and it Non-Backorder product.This would cost the company as the fact that we predicted few items as Backorder items but they were not actually in Backorder list. The warehousing cost for such items is the cost associated with the false positives.
- False Negative : A miss. The product is missed to be detected by the model and product stays as Non-Backorder.This will cost the company as cost associated with incorrectly missing items when actual demand was there for them.

F1 Score will be used when False Positives and False Negatives are more important as they are crucial for Bussiness cost.

Among F1 Score, we will choose Macro Averaged F1 Score since we would like to treat both class as equals. Micro F1 Score will be used when we want to maximise the classification of a particular class which is not the case in this problem. Due to imbalanced dataset, we will get high Micro Average F1 Score which is not called for.

Import Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.impute import KNNImputer
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
from pandas import read_csv
```

```
from numpy.random import seed
from keras.layers import Input, Dense
from keras.models import Model
from tensorflow import keras
from tensorflow.keras import layers

from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

from tensorflow.keras.layers import Dense,Input,Conv2D,MaxPool2D,Activation,Dropout,F
from tensorflow.keras.models import Model
import random as rn
from keras.models import Sequential
from keras.layers import LeakyReLU
from keras.layers import BatchNormalization
from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint

import imblearn
from imblearn.over_sampling import SMOTE
from imblearn.over_sampling import SMOTENC

from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
from sklearn.metrics import f1_score

from scipy.stats import chi2_contingency
from scipy.stats import chi2
import scipy.stats as stats
from scipy.stats import f_oneway
from scipy import stats
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

from imblearn.under_sampling import RandomUnderSampler
from xgboost import XGBClassifier

from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
import random
import pickle
from joblib import dump,load
```

```
%tensorflow version 2.x
```

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

```
Mounted at /content/drive
```

Load Dataset

```
df_train = pd.read_csv("/content/drive/MyDrive/Kaggle_Training_Dataset_v2.csv")
df_test = pd.read_csv("/content/drive/MyDrive/Kaggle_Test_Dataset_v2.csv")
```

Count rows and columns for train and test data

```
print("Number of rows in train data : ",df_train.shape[0])
print("Number of rows in test data : ",df_test.shape[0])
print("Number of columns in train data : ",df_train.shape[1])
print("Number of columns in test data : ",df_test.shape[1])
```

```
Number of rows in train data :  1687861
Number of rows in test data :  242076
Number of columns in train data :  23
Number of columns in test data :  23
```

```
# Removing the last rows of Train as well as Test data which contains Nan For all col
df_train.drop(df_train.tail(1).index,inplace=True)
df_test.drop(df_test.tail(1).index,inplace=True)
```

Merge Train and Test Data

```
df = df_train.append(df_test,ignore_index=True)
```

```
df.shape
```

```
(1929935, 23)
```

```
df.describe()
```

	national_inv	lead_time	in_transit_qty	forecast_3_month	forecast_6_mont
count	1.929935e+06	1.814318e+06	1.929935e+06	1.929935e+06	1.929935e+0
mean	4.965683e+02	7.878627e+00	4.306440e+01	1.785399e+02	3.454659e+0
std	2.957343e+04	7.054212e+00	1.295420e+03	5.108770e+03	9.831562e+0
min	-2.725600e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+0

1

Check for Unique rows

[ ] ↳ *2 cells hidden*

Check for Nan Values

[ ] ↳ *1 cell hidden*

Comment

- lead_time column has 115617 Nan values.Need to perform imputation because of this.

Count of Target values of Imbalanced dataset

[ ] ↳ *5 cells hidden*

Exploratory Data Analysis

[ ] ↳ *114 cells hidden*

Finding Correlation between 2 Categorical Features

[ ] ↳ *5 cells hidden*

Finding Correlation between Categorical and Numerical Feature using Point Biserial Correlation

The Point Biserial Correlation is used to measure the correlation between a Categorical Variable(Binary Category) and Continous Variable

The Correlation coeffecients varies between -1 to +1 with 0 implying No Correlation.

Null Hypothesis : There is no correlation between the two features.Features.

Alternate Hypothesis : There is a correlation between the two features.

We can verify the hypothesis using **p-value**.

[ ] ↳ *7 cells hidden*

Data Preprocessing

Converting Target Variable to 0 and 1

```
#Converting Target Variable to 0 and 1
df['went_on_backorder'].replace({'Yes':1,'No':0},inplace=True)
df['went_on_backorder'].astype(int)
```

```
0          0
1          0
2          0
3          0
4          0
          ..
1929930    0
1929931    0
1929932    0
1929933    0
1929934    0
Name: went_on_backorder, Length: 1929935, dtype: int64
```

Check for Number of 0s in Numerical Features and drop columns if 0s are more than 95%.

```
# Drop Columns if 95% values are 0
zero_columns = []
for col in df.columns:
    if col!='went_on_backorder' and col!='sku':
        value = df[col].value_counts()
        if 0 in value:
            if value[0] >= (0.95*df.shape[0]):
                zero_columns.append(col)
                print(col,"-> percentage of zeros : ",(value[0]*100/df.shape[0]),"%")
```

```
pieces_past_due -> percentage of zeros :  98.53347392528764 %
local_bo_qty -> percentage of zeros :  98.64223406487783 %
```

```
df = df.drop(columns=zero_columns,axis=1)
```

Replacing -99 by Nan in performance column

```
# replacing -99 by Nan in performance column
df.perf_6_month_avg.replace({-99.0 : np.nan},inplace=True)
df.perf_12_month_avg.replace({-99.0 : np.nan},inplace=True)
```

Converting Categorical Features

```
categorical_columns = ['rev_stop','stop_auto_buy','ppap_risk','oe_constraint','deck_r
for col in categorical_columns:
    df[col].replace({'Yes':1,'No':0},inplace=True)
    df[col]=df[col].astype(int)
```

Removing Outlier Datapoints from DataFrame

```
old_shape = df.shape[0]
```

```
df = df[(df.national_inv >= 0.000) & (df.national_inv <= 5487.000) & (df.in_transit_q
  (df.forecast_3_month <= 2280.000) & (df.forecast_6_month <= 4335.659999999916) &\
  (df.forecast_9_month <= 6316.000) & (df.sales_1_month <= 693.000) & (df.sales_3_mon
  (df.sales_6_month <= 4410.000) & (df.sales_9_month <= 6698.000) & (df.min_bank <= 6
```

```
new_shape = df.shape[0]
```

```
print("Number of Outliers removed : ",old_shape-new_shape)
```

```
Number of Outliers removed :  48564
```

```
df.describe()
```

	national_inv	lead_time	in_transit_qty	forecast_3_month	forecast_6_montl
count	1.881371e+06	1.768920e+06	1.881371e+06	1.881371e+06	1.881371e+0(
mean	1.376750e+02	7.879902e+00	1.145663e+01	3.950405e+01	7.661432e+0
std	4.669358e+02	7.043846e+00	7.915361e+01	1.639866e+02	3.039033e+0
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+0(
25%	4.000000e+00	4.000000e+00	0.000000e+00	0.000000e+00	0.000000e+0(
50%	1.400000e+01	8.000000e+00	0.000000e+00	0.000000e+00	0.000000e+0(
75%	7.200000e+01	9.000000e+00	0.000000e+00	3.000000e+00	1.000000e+0
max	5.487000e+03	5.200000e+01	5.300000e+03	2.280000e+03	4.320000e+0

Train Test Split

```
# Assigning the Target Variable Column to y_true variable and droping it from the DF
y_true = df['went_on_backorder']
```

```
df = df.drop(['sku','went_on_backorder'],axis=1)
```

```
X_train,X_test,y_train,y_test = train_test_split(df,y_true,stratify=y_true,test_size=
X_train,X_cv,y_train,y_cv = train_test_split(X_train,y_train,stratify=y_train,test_si
print(X_train.shape,X_cv.shape,X_test.shape)
```

```
(1204076, 19) (301020, 19) (376275, 19)
```

Median Imputation

```
median_values = X_train.median()

X_train_median = X_train.fillna(median_values)
X_cv_median = X_cv.fillna(median_values)
X_test_median = X_test.fillna(median_values)

print(X_train_median.shape,X_cv_median.shape,X_test_median.shape)
```

```
(1204076, 19) (301020, 19) (376275, 19)
```

Feature Engineering

Adding SVD to Median Imputation

```
Trunc_SVD = TruncatedSVD(n_components=2,n_iter=20)
X_train_median_SVD = Trunc_SVD.fit_transform(X_train_median)
X_cv_median_SVD = Trunc_SVD.transform(X_cv_median)
X_test_median_SVD = Trunc_SVD.transform(X_test_median)
print(X_train_median_SVD.shape,X_cv_median.shape,X_test_median_SVD.shape)
```

```
(1204076, 2) (301020, 19) (376275, 2)
```

AutoEncoder for Feature Engineering

```
tf.keras.backend.clear_session()
np.random.seed(0)
rn.seed(0)
```

```
earlystop = EarlyStopping(monitor='val_loss', min_delta=0.1, patience=3, verbose=1)
reduce_lr_on_val_loss = ReduceLROnPlateau(monitor='val_loss' factor=0.9 patience=1 ve
```

```
reduce_lr_on_val_loss = ReduceLROnPlateau(monitor='val_loss',factor=0.9,patience=1,ve

filepath="model_save_median/weights-{epoch:02d}-{val_loss:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_loss', verbose=1, save_b
```

```
#https://machinelearningmastery.com/autoencoder-for-classification/
number_of_columns = X_train_median.shape[1]
encoder_input_dim = Input(shape=(number_of_columns,))

#encoder level 1
e = Dense(number_of_columns)(encoder_input_dim)
e = BatchNormalization()(e)
e = LeakyReLU()(e)

# encoder level 2
e = Dense(10)(e)
e = BatchNormalization()(e)
e = LeakyReLU()(e)

#Bottleneck
n_bottleneck = 2
bottleneck = Dense(n_bottleneck)(e)
```

```
d = Dense(10)(bottleneck)
d = BatchNormalization()(d)
d = LeakyReLU()(d)

# decoder level 2
d = Dense(number_of_columns)(d)
d = BatchNormalization()(d)
d = LeakyReLU()(d)
# output
encoder_output = Dense(number_of_columns,activation='linear')(d)
# Defining auto encoder model
model = Model(inputs=encoder_input_dim,outputs = encoder_output)

callback_list = [earlystop,reduce_lr_on_val_loss,checkpoint]
model.compile(optimizer='adam',loss='mse')
model.fit(X_train_median,y_train,epochs=25,batch_size=100,shuffle=True,validation_dat

```

```
Epoch 1/25
12041/12041 [==============================] - 37s 3ms/step - loss: 0.6548 - val_loss

Epoch 00001: val_loss improved from inf to 0.00688, saving model to model_save_mediar
Epoch 2/25
12041/12041 [==============================] - 34s 3ms/step - loss: 0.0069 - val_loss

Epoch 00002: val_loss improved from 0.00688 to 0.00643, saving model to model_save_me
Epoch 3/25
12041/12041 [==============================] - 34s 3ms/step - loss: 0.0063 - val_loss

Epoch 00003: ReduceLROnPlateau reducing learning rate to 0.0009000000427477062.

Epoch 00003: val_loss did not improve from 0.00643
```

```
Epoch 4/25
12041/12041 [==============================] - 34s 3ms/step - loss: 0.0062 - val_loss

Epoch 00004: val_loss improved from 0.00643 to 0.00612, saving model to model_save_me
Epoch 00004: early stopping
<tensorflow.python.keras.callbacks.History at 0x7f6338198790>
```

```
encoder = Model(inputs =encoder_input_dim,outputs = bottleneck)
encoder.save('encoder.h5')
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be
```

```
encoded_output_Train = encoder.predict(X_train_median)
encoded_output_cv = encoder.predict(X_cv_median)
encoded_output_test = encoder.predict(X_test_median)
```

```
print(encoded_output_Train.shape,encoded_output_cv.shape,encoded_output_test.shape)
```

```
(1204076, 2) (301020, 2) (376275, 2)
```

Feature Binning using Decision Tree

[ ] ↳ *11 cells hidden*

Adding the Feature Engineered Data to the Main DataFrame

[ ] ↳ *5 cells hidden*

Over Sampling Technique : SMOTE-NC

[ ] ↳ *4 cells hidden*

Normalise All Columns after SMOTE

[ ] ↳ *1 cell hidden*

Under Sampling

[ ] ↳ *7 cells hidden*

Key points and takeways from EDA and Feature Engineering

- We are solving binary classification problem with very high data imbalance.Positive class being the majority.
- Categorical features consists of Yes and No Category.
- SKU is unique and is not important for classification and hence will be dropped.
- All numerical features had extreme skewness on Right indicating them as Outliers.
- in Lead_time feature there are many Nan values. performed imputations on that feature.
- Performance average features had -99 as their value.implemented Median imputation.
- From the Correlation Matrix The Forecast,performance average and sales features are extremely correlated to each other.
- Performed Chi Square test on 2 categorical features. Made Null hypothesis for this correlation and performed statistical tests.Results state that there is no correlation between rev_stop feature and the target variable.
- Performed Point Biserial correlation test to check for correlation between categorical and numerical features.Forcast features , sales 1 month and national_inv feature are not correlated to the target variable.
- local_bo_qty and piecies_past_due Features have close to 98% of 0s in them. They have been removed.
- Removed close to 48k outlier datapoints as a part of data cleaning process.
- Performed SMOTE-NC as we have categorical data as well , an oversampling technique to increase minority class duplicate datapoints to increase percentage of minority class.
- Used RandomUnderSampler as an undersampling technique to reduce the datapoints from majority class.
- Apart from given features, also added 2-2 Truncated SVD features, AutoEncoder Features,Discretised bin features as a feature extraction method.

Random Model

[ ] ↳ *1 cell hidden*

Machine Learning Models

Logistic Regression

[ ] ↳ *15 cells hidden*

Random Forest

[ ] ↳ *17 cells hidden*

XGBOOST

[ ] ↳ 31 cells hidden

Ensemble Model

Without Feature Engineering

```
# Assigning the Target Variable Column to y_true variable and droping it from the DF
y_true = df['went_on_backorder']
```

```
df = df.drop(['sku','went_on_backorder'],axis=1)
```

```
df.shape
```

```
(1881371, 19)
```

```
X_train_en,X_test_en,y_train_en,y_test_en=train_test_split(df,y_true,stratify=y_true,
print(X_train_en.shape,X_test_en.shape,y_train_en.shape,y_test_en.shape)
```

```
(1505096, 19) (376275, 19) (1505096,) (376275,)
```

```
data_d1,data_d2,y_d1,y_d2 = train_test_split(X_train_en,y_train_en,stratify=y_train_e
print(data_d1.shape,data_d2.shape,y_d1.shape,y_d2.shape)
```

```
(752548, 19) (752548, 19) (752548,) (752548,)
```

```
median_values = data_d1.median()

columns = data_d1.columns

data_d1 = data_d1.fillna(median_values)
data_d2 = data_d2.fillna(median_values)
X_test_en = X_test_en.fillna(median_values)

print(data_d1.shape,data_d2.shape,X_test_en.shape)
```

```
(752548, 19) (752548, 19) (376275, 19)
```

```
data_d1 = pd.DataFrame(data_d1,columns=columns)
data_d2 = pd.DataFrame(data_d2,columns=columns)
X_test_en = pd.DataFrame(X_test_en,columns=columns)
```

```
# Normalise Median Imputed Data
MinMaxSc = MinMaxScaler()
```

```

columns = data_d1.columns
data_d1 = pd.DataFrame(MinMaxSc.fit_transform(data_d1),columns=columns)
data_d2 = pd.DataFrame(MinMaxSc.transform(data_d2),columns=columns)
X_test_en = pd.DataFrame(MinMaxSc.transform(X_test_en),columns=columns)

print(data_d1.shape,data_d2.shape,X_test_en.shape)
```

```
(752548, 19) (752548, 19) (376275, 19)
```

```
def generating_samples_with_replacement(input_data,output_data):
    rows_selected = random.sample(range(0,len(input_data)-1),int(0.6*(len(input_data)
    sampled_input = input_data.iloc[rows_selected]
    sampled_output = output_data.iloc[rows_selected]

    sampled_input = sampled_input.values.tolist()
    sampled_output = sampled_output.values.tolist()

    replicate = random.sample(range(0,len(sampled_input)-1),int(0.4*(len(input_data))
    for num in replicate:
        sampled_input.append(sampled_input[num])
        sampled_output.append(sampled_output[num])

    return sampled_input,sampled_output
```

```
def baseModel_DecisionTree(number_of_models,data_d1,y_d1,data_d2,y_d2):
    input_data_list=[]
    output_data_list=[]
    clf_list=[]
    output_list=[]
    final_data = []

    print("Generating samples with 60% sample and 40% sample with replacement")
    print("In Sample Generate")
    for num in range(number_of_models):
        print("Sample generation for Model : ",num)
        inp,outp = generating_samples_with_replacement(data_d1,y_d1)
        input_data_list.append(inp)
        output_data_list.append(outp)
    print("Out Sample Generate")

    print("Fit Decision Tree Models on samples generated")
    print("In Fitting Models")
    for num in range(number_of_models):
        print("Fitting for Model : ",num)
        DT = DecisionTreeRegressor(max_depth=None)
        clf = DT.fit(input_data_list[num],output_data_list[num])
        clf_list.append(clf)
    print("Out Fitting Models")

    print("Predict values for second dataset")
    print("In Predict")
    for num in range(number_of_models):
        print("Predict for Model : ",num)
```

```
        clf = clf_list[num]
        output = clf.predict(data_d2)
        output_list.append(output)
    print("Out Predict")

    print("Zip each row of the column to form dataset")
    print(" In form Meta Data")
    for num in range(len(output_list[0])):
        output=[]
        for i in range(number_of_models):
            output.append(output_list[i][num])
        final_data.append(output)
    print("Out form Meta Data")

    final_data = pd.DataFrame(final_data)

    return clf_list,final_data
```

```
number_of_models=15
modelList,data = baseModel_DecisionTree(number_of_models,data_d1,y_d1,data_d2,y_d2)
```

```
Generating samples with 60% sample and 40% sample with replacement
In Sample Generate
Sample generation for Model :  0
Sample generation for Model :  1
Sample generation for Model :  2
Sample generation for Model :  3
Sample generation for Model :  4
Sample generation for Model :  5
Sample generation for Model :  6
Sample generation for Model :  7
Sample generation for Model :  8
Sample generation for Model :  9
Sample generation for Model :  10
Sample generation for Model :  11
Sample generation for Model :  12
Sample generation for Model :  13
Sample generation for Model :  14
Out Sample Generate
Fit Decision Tree Models on samples generated
In Fitting Models
Fitting for Model :  0
Fitting for Model :  1
Fitting for Model :  2
Fitting for Model :  3
Fitting for Model :  4
Fitting for Model :  5
Fitting for Model :  6
Fitting for Model :  7
Fitting for Model :  8
Fitting for Model :  9
Fitting for Model :  10
Fitting for Model :  11
Fitting for Model :  12
Fitting for Model :  13
Fitting for Model :  14
Out Fitting Models
Predict values for second dataset
```

```
In Predict
Predict for Model :  0
Predict for Model :  1
Predict for Model :  2
Predict for Model :  3
Predict for Model :  4
Predict for Model :  5
Predict for Model :  6
Predict for Model :  7
Predict for Model :  8
Predict for Model :  9
Predict for Model :  10
Predict for Model :  11
Predict for Model :  12
Predict for Model :  13
Predict for Model :  14
Out Predict
Zip each row of the column to form dataset
 In form Meta Data
Out form Meta Data
```

```
# Meta Classifier - Logistic Regression
```

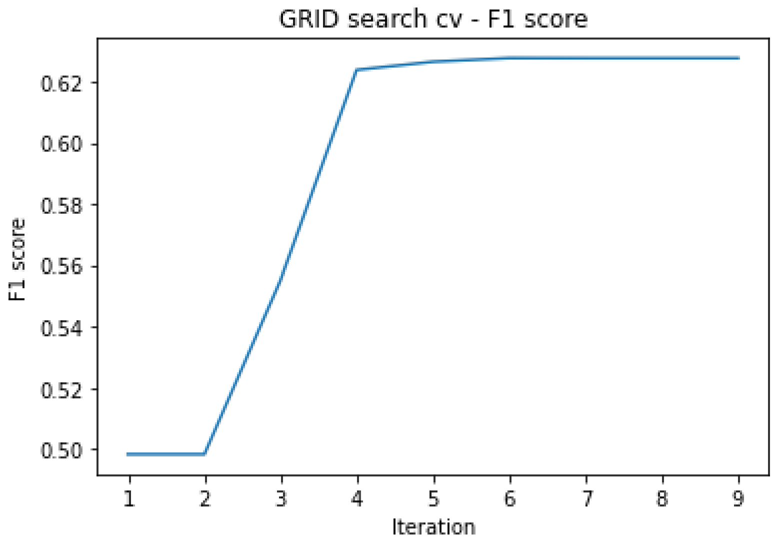
```
lg_clf = LogisticRegression()
params = {'C':[10**x for x in range(-5,4)]}
grid_log_clf = GridSearchCV(lg_clf,param_grid=params,n_jobs=-1,scoring='f1_macro',ver
grid_log_clf.fit(data,y_d2.values.ravel())
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed:   27.9s
[Parallel(n_jobs=-1)]: Done  45 out of  45 | elapsed:   45.9s finished
GridSearchCV(cv=None, error_score=nan,
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='auto',
                                          n_jobs=None, penalty='l2',
                                          random_state=None, solver='lbfgs',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'C': [1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100,
                               1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='f1_macro', verbose=3)
```

```
labels = grid_log_clf.cv_results_['params']
x_axis = range(1,10)
y_axis = grid_log_clf.cv_results_['mean_test_score']
for i,label in enumerate(labels):
    print(label,":",y_axis[i])

plt.xlabel("Iteration")
plt.ylabel("F1 score")
plt.title("GRID search cv - F1 score")
plt.plot(x_axis,y_axis)
```

```
{'C': 1e-05} : 0.4982939749796286
{'C': 0.0001} : 0.4982939749796286
{'C': 0.001} : 0.5551692329818663
{'C': 0.01} : 0.6238061616131717
{'C': 0.1} : 0.6264931744337635
{'C': 1} : 0.6276808492590912
{'C': 10} : 0.6276425035563356
{'C': 100} : 0.6276425035563356
{'C': 1000} : 0.6276425035563356
[<matplotlib.lines.Line2D at 0x7f47a58af4d0>]
```




```
grid_log_clf.best_estimator_
```

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
clf_best = LogisticRegression(C=1)
clf_best.fit(data,y_d2.values.ravel())
```

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
def perform_test(number_of_models,clf_list,X_test_en):
  test_output_list=[]
  test_data = []

  print("In Predicting Output for Test Data")
  for num in range(number_of_models):
    clf = clf_list[num]
    output = clf.predict(X_test_en)
    test_output_list.append(output)
  print("Out Predicting Output for Test Data")

  for i in range(len(test_output_list[0])):
```

```
      outp=[]
      for j in range(number_of_models):
        outp.append(test_output_list[j][i])
      test_data.append(outp)
    test_data = pd.DataFrame(test_data)

    return test_data
```

```
test_data = perform_test(number_of_models,modelList,X_test_en)
y_test_pred = clf_best.predict(test_data)
```

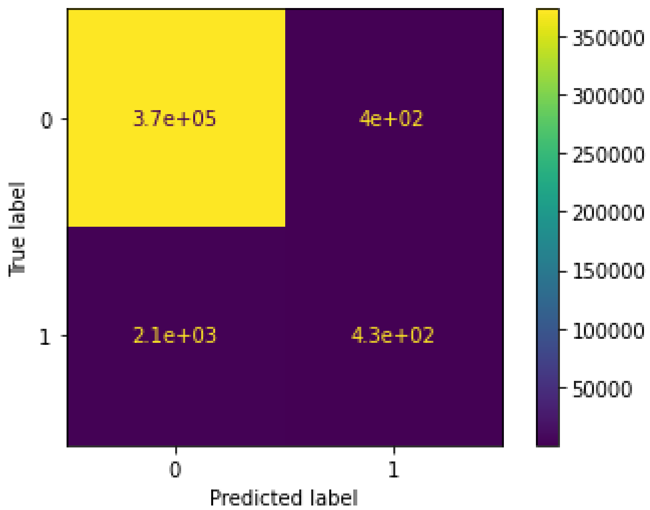
```
In Predicting Output for Test Data
Out Predicting Output for Test Data
```

```
print("Test F1 score : ",f1_score(y_test_en,y_test_pred,average='macro'))
```

```
Test F1 score :  0.6361531436954092
```

```
confusionMatrix = confusion_matrix(y_test_en,y_test_pred,labels=[0,1])
cm_display = ConfusionMatrixDisplay(confusion_matrix = confusionMatrix,display_labels
cm_display.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f2c03eb9a90>
```



With Feature Engineering

```
# Assigning the Target Variable Column to y_true variable and droping it from the DF
y_true = df['went_on_backorder']
```

```
df = df.drop(['sku','went_on_backorder'],axis=1)
```

```
df.shape
```

```
(1881371, 19)
```

```
X_train_en,X_test_en,y_train_en,y_test_en=train_test_split(df,y_true,stratify=y_true,
print(X_train_en.shape,X_test_en.shape,y_train_en.shape,y_test_en.shape)
```

```
(1505096, 19) (376275, 19) (1505096,) (376275,)
```

```
data_d1,data_d2,y_d1,y_d2 = train_test_split(X_train_en,y_train_en,stratify=y_train_e
print(data_d1.shape,data_d2.shape,y_d1.shape,y_d2.shape)
```

```
(752548, 19) (752548, 19) (752548,) (752548,)
```

```
median_values = data_d1.median()

columns = data_d1.columns

data_d1 = data_d1.fillna(median_values)
data_d2 = data_d2.fillna(median_values)
X_test_en = X_test_en.fillna(median_values)

print(data_d1.shape,data_d2.shape,X_test_en.shape)
```

```
(752548, 19) (752548, 19) (376275, 19)
```

```
data_d1 = pd.DataFrame(data_d1,columns=columns)
data_d2 = pd.DataFrame(data_d2,columns=columns)
X_test_en = pd.DataFrame(X_test_en,columns=columns)
```

SVD

```
Trunc_SVD = TruncatedSVD(n_components=2,n_iter=20)
data_d1_SVD = Trunc_SVD.fit_transform(data_d1)
data_d2_SVD = Trunc_SVD.transform(data_d2)
X_test_en_SVD = Trunc_SVD.transform(X_test_en)
print(data_d1_SVD.shape,data_d2_SVD.shape,X_test_en_SVD.shape)
```

```
(752548, 2) (752548, 2) (376275, 2)
```

Auto Encoders

```
tf.keras.backend.clear_session()
np.random.seed(0)
rn.seed(0)
```

```
earlystop = EarlyStopping(monitor='val_loss', min_delta=0.1, patience=3, verbose=1)
reduce_lr_on_val_loss = ReduceLROnPlateau(monitor='val_loss',factor=0.9,patience=1,ve

filepath="model_save_median/weights-{epoch:02d}-{val_loss:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_loss', verbose=1, save_b
```

```
#https://machinelearningmastery.com/autoencoder-for-classification/
number_of_columns = data_d1.shape[1]
encoder_input_dim = Input(shape=(number_of_columns,))

#encoder level 1
e = Dense(number_of_columns)(encoder_input_dim)
e = BatchNormalization()(e)
e = LeakyReLU()(e)

# encoder level 2
e = Dense(10)(e)
e = BatchNormalization()(e)
e = LeakyReLU()(e)

#Bottleneck
n_bottleneck = 2
bottleneck = Dense(n_bottleneck)(e)
```

```
d = Dense(10)(bottleneck)
d = BatchNormalization()(d)
d = LeakyReLU()(d)

# decoder level 2
d = Dense(number_of_columns)(d)
d = BatchNormalization()(d)
d = LeakyReLU()(d)
# output
encoder_output = Dense(number_of_columns,activation='linear')(d)
# Defining auto encoder model
model = Model(inputs=encoder_input_dim,outputs = encoder_output)

callback_list = [earlystop,reduce_lr_on_val_loss,checkpoint]
model.compile(optimizer='adam',loss='mse')
model.fit(data_d1,y_d1,epochs=25,batch_size=100,shuffle=True,validation_data=(data_d2

```

```
Epoch 1/25
7526/7526 [==============================] - 28s 3ms/step - loss: 0.6895 - val_loss:

Epoch 00001: val_loss improved from inf to 0.00816, saving model to model_save_mediar
Epoch 2/25
7526/7526 [==============================] - 27s 4ms/step - loss: 0.0079 - val_loss:

Epoch 00002: val_loss improved from 0.00816 to 0.00681, saving model to model_save_me
Epoch 3/25
7526/7526 [==============================] - 24s 3ms/step - loss: 0.0075 - val_loss:

Epoch 00003: ReduceLROnPlateau reducing learning rate to 0.0009000000427477062.

Epoch 00003: val_loss improved from 0.00681 to 0.00676, saving model to model_save_me
Epoch 4/25
7526/7526 [==============================] - 25s 3ms/step - loss: 0.0067 - val_loss:

Epoch 00004: val_loss improved from 0.00676 to 0.00669, saving model to model_save_me
```

```
Epoch 00004: early stopping
<tensorflow.python.keras.callbacks.History at 0x7f3a3fde1090>
```

```
encoder = Model(inputs =encoder_input_dim,outputs = bottleneck)
encoder.save('encoder.h5')
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be
```

```
encoded_output_data_d1 = encoder.predict(data_d1)
encoded_output_data_d2 = encoder.predict(data_d2)
encoded_output_test = encoder.predict(X_test_en)
```

```
print(encoded_output_data_d1.shape,encoded_output_data_d2.shape,encoded_output_test.s
```

```
(752548, 2) (752548, 2) (376275, 2)
```

Feature Binning

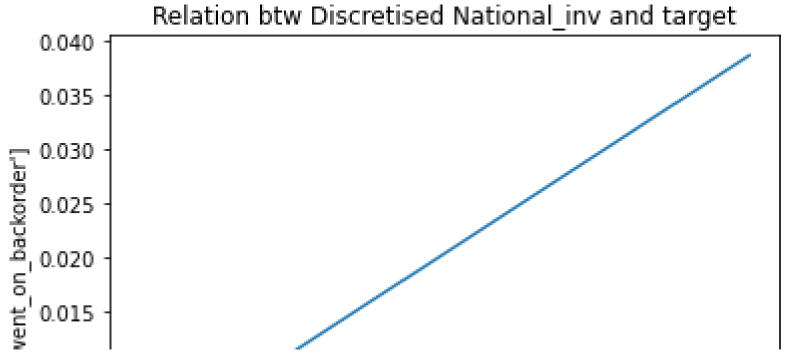
```
# Bins and Probabilities for National_inv Feature
clf_inv = DecisionTreeClassifier(max_depth=2)
clf_inv.fit(data_d1.national_inv.to_frame(),y_d1)
data_d1['national_inv_prob'] = clf_inv.predict_proba(data_d1.national_inv.to_frame())
print(data_d1.national_inv_prob.head(10))
```

```
844820     0.007317
197031     0.002152
897553     0.002152
1399467    0.002152
1451080    0.007317
178399     0.002152
488875     0.007317
798025     0.007317
839330     0.002152
694483     0.002152
Name: national_inv_prob, dtype: float64
```

```
# Adding target variable to check relation
data_d1['went_on_backorder'] = y_d1
```

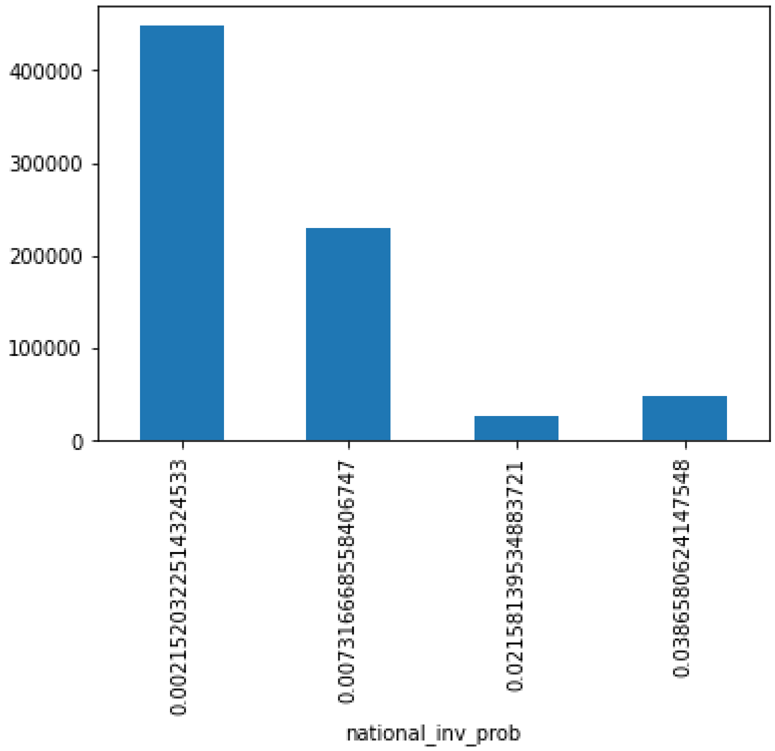
```
fig = plt.figure()
fig = data_d1.groupby(['national_inv_prob'])['went_on_backorder'].mean().plot()
fig.set_title("Relation btw Discretised National_inv and target")
fig.set_ylabel(['went_on_backorder'])
```

```
Text(0, 0.5, "['went_on_backorder']")
```




```
#check the number of products per probabilistic bin to understand distribution of dis
data_d1.groupby(['national_inv_prob'])['went_on_backorder'].count().plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f39ec101410>
```




```
print("Unique values of National_inv prob : ",data_d1.national_inv_prob.unique())
```

```
Unique values of National_inv prob :  [0.00731667 0.00215203 0.03865806 0.0215814 ]
```

```
pd.concat([data_d1.groupby(['national_inv_prob'])['national_inv'].min(),
           data_d1.groupby(['national_inv_prob'])['national_inv'].max()],axis=1)
```

national_inv_prob	national_inv	national_inv
0.002152	10.0	5486.0
0.007317	2.0	9.0
0.021581	1.0	1.0
0.038658	0.0	0.0

```
data_d2['national_inv_prob'] = clf_inv.predict_proba(data_d2.national_inv.to_frame())
X_test_en['national_inv_prob'] = clf_inv.predict_proba(X_test_en.national_inv.to_fram
```

```
data_d1['national_inv_bins'] = 0
data_d1.loc[(data_d1['national_inv'] == 0.0),'national_inv_bins'] = 1
data_d1.loc[(data_d1['national_inv'] == 1.0),'national_inv_bins'] = 2
data_d1.loc[(data_d1['national_inv'] >= 2.0) & (data_d1['national_inv'] <= 9.0),'nati
data_d1.loc[(data_d1['national_inv'] >= 10.0),'national_inv_bins'] = 4

data_d2['national_inv_bins'] = 0
data_d2.loc[(data_d2['national_inv'] == 0.0),'national_inv_bins'] = 1
data_d2.loc[(data_d2['national_inv'] == 1.0),'national_inv_bins'] = 2
data_d2.loc[(data_d2['national_inv'] >= 2.0) & (data_d2['national_inv'] <= 9.0),'nati
data_d2.loc[(data_d2['national_inv'] >= 10.0),'national_inv_bins'] = 4

X_test_en['national_inv_bins'] = 0
X_test_en.loc[(X_test_en['national_inv'] == 0.0),'national_inv_bins'] = 1
X_test_en.loc[(X_test_en['national_inv'] == 1.0),'national_inv_bins'] = 2
X_test_en.loc[(X_test_en['national_inv'] >= 2.0) & (X_test_en['national_inv'] <= 9.0)
X_test_en.loc[(X_test_en['national_inv'] >= 10.0),'national_inv_bins'] = 4
```

```
data_d1 = data_d1.drop(['went_on_backorder'],axis=1)
```

Adding SVD and AutoEncoder Features in the main data

```
for i in range(2):
    data_d1['T_SVD_'+str(i)] = data_d1_SVD[:,i]
    data_d2['T_SVD_'+str(i)] = data_d2_SVD[:,i]
    X_test_en['T_SVD_'+str(i)] = X_test_en_SVD[:,i]
print(data_d1.shape,data_d2.shape,X_test_en.shape)
```

```
(752548, 23) (752548, 23) (376275, 23)
```

```
for i in range(2):
    data_d1['AutoEncoder_'+str(i)] = encoded_output_data_d1[:,i]
    data_d2['AutoEncoder_'+str(i)] = encoded_output_data_d2[:,i]
    X_test_en['AutoEncoder_'+str(i)] = encoded_output_test[:,i]
print(data_d1.shape,data_d2.shape,X_test_en.shape)
```

```
(752548, 25) (752548, 25) (376275, 25)
```

Normalise Data

```
# Normalise Median Imputed Data
MinMaxSc = MinMaxScaler()

columns = data_d1.columns
data_d1 = pd.DataFrame(MinMaxSc.fit_transform(data_d1),columns=columns)
data_d2 = pd.DataFrame(MinMaxSc.transform(data_d2),columns=columns)
```

```
data_d2 = pd.DataFrame(MinMaxSc.transform(data_d2),columns=columns)
X_test_en = pd.DataFrame(MinMaxSc.transform(X_test_en),columns=columns)

print(data_d1.shape,data_d2.shape,X_test_en.shape)
```

```
(752548, 25) (752548, 25) (376275, 25)
```

```
number_of_models=15
modelList,data = baseModel_DecisionTree(number_of_models,data_d1,y_d1,data_d2,y_d2)
```

```
Generating samples with 60% sample and 40% sample with replacement
In Sample Generate
Sample generation for Model :  0
Sample generation for Model :  1
Sample generation for Model :  2
Sample generation for Model :  3
Sample generation for Model :  4
Sample generation for Model :  5
Sample generation for Model :  6
Sample generation for Model :  7
Sample generation for Model :  8
Sample generation for Model :  9
Sample generation for Model :  10
Sample generation for Model :  11
Sample generation for Model :  12
Sample generation for Model :  13
Sample generation for Model :  14
Out Sample Generate
Fit Decision Tree Models on samples generated
In Fitting Models
Fitting for Model :  0
Fitting for Model :  1
Fitting for Model :  2
Fitting for Model :  3
Fitting for Model :  4
Fitting for Model :  5
Fitting for Model :  6
Fitting for Model :  7
Fitting for Model :  8
Fitting for Model :  9
Fitting for Model :  10
Fitting for Model :  11
Fitting for Model :  12
Fitting for Model :  13
Fitting for Model :  14
Out Fitting Models
Predict values for second dataset
In Predict
Predict for Model :  0
Predict for Model :  1
Predict for Model :  2
Predict for Model :  3
Predict for Model :  4
Predict for Model :  5
Predict for Model :  6
Predict for Model :  7
Predict for Model :  8
Predict for Model :  9
Predict for Model :  10
Predict for Model :  11
```

```
Predict for Model :  12
Predict for Model :  13
Predict for Model :  14
Out Predict
Zip each row of the column to form dataset
 In form Meta Data
Out form Meta Data
```

```
# Meta Classifier - Logistic Regression
```

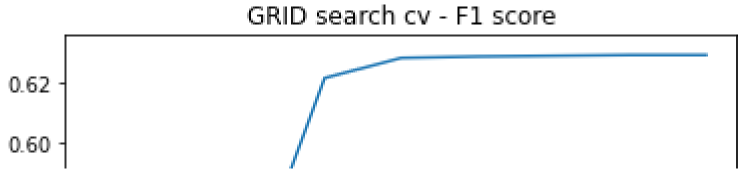
```
lg_clf = LogisticRegression()
params = {'C':[10**x for x in range(-5,4)]}
grid_log_clf = GridSearchCV(lg_clf,param_grid=params,n_jobs=-1,scoring='f1_macro',ver
grid_log_clf.fit(data,y_d2.values.ravel())
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed:   24.3s
[Parallel(n_jobs=-1)]: Done  45 out of  45 | elapsed:   39.4s finished
GridSearchCV(cv=None, error_score=nan,
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='auto',
                                          n_jobs=None, penalty='l2',
                                          random_state=None, solver='lbfgs',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'C': [1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100,
                               1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='f1_macro', verbose=3)
```

```
labels = grid_log_clf.cv_results_['params']
x_axis = range(1,10)
y_axis = grid_log_clf.cv_results_['mean_test_score']
for i,label in enumerate(labels):
    print(label,":",y_axis[i])

plt.xlabel("Iteration")
plt.ylabel("F1 score")
plt.title("GRID search cv - F1 score")
plt.plot(x_axis,y_axis)
```

```
{'C': 1e-05} : 0.4982939749796286
{'C': 0.0001} : 0.4982939749796286
{'C': 0.001} : 0.5534706671930081
{'C': 0.01} : 0.621628415859751
{'C': 0.1} : 0.6283068586396076
{'C': 1} : 0.6287836210301228
{'C': 10} : 0.6290223855418355
{'C': 100} : 0.6292815696831074
{'C': 1000} : 0.6292815696831074
[<matplotlib.lines.Line2D at 0x7f3a3017db50>]
```




```
grid_log_clf.best_estimator_
```

```
LogisticRegression(C=100, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
clf_best = LogisticRegression(C=10)
clf_best.fit(data,y_d2.values.ravel())
```

```
LogisticRegression(C=10, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
test_data = perform_test(number_of_models,modelList,X_test_en)
y_test_pred = clf_best.predict(test_data)
```

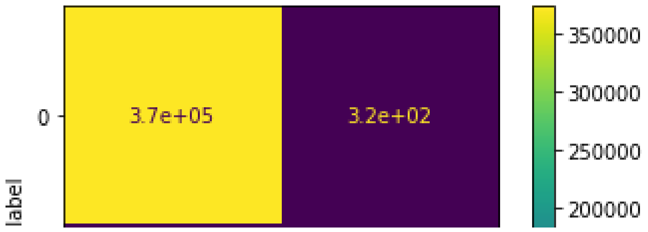
```
In Predicting Output for Test Data
Out Predicting Output for Test Data
```

```
print("Test F1 score : ",f1_score(y_test_en,y_test_pred,average='macro'))
```

```
Test F1 score :  0.6207567400765969
```

```
confusionMatrix = confusion_matrix(y_test_en,y_test_pred,labels=[0,1])
cm_display = ConfusionMatrixDisplay(confusion_matrix = confusionMatrix,display_labels
cm_display.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f3a30049a10>
```



- With use of Custom Ensemble models we see that with increasing number of base models ie Decision Tree the F1 scores initially increase and then tapers off.
- With Custom Ensembles applied on with and without feature engineering data , the f1 score almost in the same range.
- Amongst all the models applied Random Forest with Oversampling data has given the best F1 score of 0.7 on cv and test data.

Summary

- We observe that with every model the improvement in f1 scores.
- We saw that the Model on oversampled data perform well as compared to Undersampled data,this could be because of loss of data while undersampling.
- Among all the models , Random Forest Model provided a f1 score of 0.69 and models including XGBoost and Custom ensembles gave a f1 score of 0.62.

```
from prettytable import PrettyTable
table = PrettyTable()
table.field_names = ['Model','Parameter','Technique','Train F1 Score','CV F1 Score','
table.add_row(['Logistic Regression','Best Alpha = '+ str(10),'SMOTE',0.79,0.50,0.50]
table.add_row(['Logistic Regression','Best Alpha = '+ str(1e-05),'RUS',0.52,0.49,0.49
table.add_row(['Random Forest',"n_estimators = "+str(500) + " , max_depth = "+str(50)
table.add_row(['Random Forest',"n_estimators = "+str(100) + " , max_depth = "+str(20)
table.add_row(['XGBOOST',"n_estimators = "+str(500) + " , learning rate = "+str(0.1)
table.add_row(['XGBOOST',"n_estimators = "+str(500) + " , learning rate = "+str(0.2)
table.add_row(['AdaBoost',"n_estimators = "+str(500) + " , learning rate = "+str(1),'
table.add_row(['AdaBoost',"n_estimators = "+str(500) + " , learning rate = "+str(0.1)
table.add_row(['Ensemble Models',"Decision Tree + Logistic Regression ",'With FE',0.6
table.add_row(['Ensemble Models',"Decision Tree + Logistic Regression ",'Without FE',
print(table)
```

```
+---------------------+---------------------------------------------------------------
|        Model        |                           Parameter
+---------------------+---------------------------------------------------------------
| Logistic Regression |                       Best Alpha = 10
| Logistic Regression |                      Best Alpha = 1e-05
|    Random Forest    | n_estimators = 500 , max_depth = 50 , min_samples_split = 5
|    Random Forest    | n_estimators = 100 , max_depth = 20 , min_samples_split = 2
|       XGBOOST       |   n_estimators = 500 , learning rate = 0.1 , max_depth = 3
```

```
|       XGBOOST       |  n_estimators = 500 , learning rate = 0.2 , max_depth = 10
|       AdaBoost      |          n_estimators = 500 , learning rate = 1
|       AdaBoost      |         n_estimators = 500 , learning rate = 0.1
|   Ensemble Models   |           Decision Tree + Logistic Regression
|   Ensemble Models   |           Decision Tree + Logistic Regression
+---------------------+---------------------------------------------------------------
```

1

1

1

1

1

1

1

1

1