SUMMER TRAINING

on

Machine Learning

Submitted by

K BHAWANA

Registration No: **11913657**

Program Name: MCA

School of Computer Application

Lovely Professional University, Phagwara

(01/MAY/2020 to 30/JUL/2020)

Acknowledgment

The training opportunity I had with Datacamp was a great chance for learning and professional development. Therefore, I consider myself as a very lucky individual as I was provided with an opportunity to be a part of it. I am also grateful for having a chance to learn from professionals who led me through this internship period.

I express my deepest thanks to Training and Placement Coordinator, School of Computer Application, Lovely Professional University for allowing me to grab this opportunity. I choose this moment to acknowledge his contribution gratefully by giving necessary advices and guidance to make my internship a good learning experience.

K Bhawana
11913657

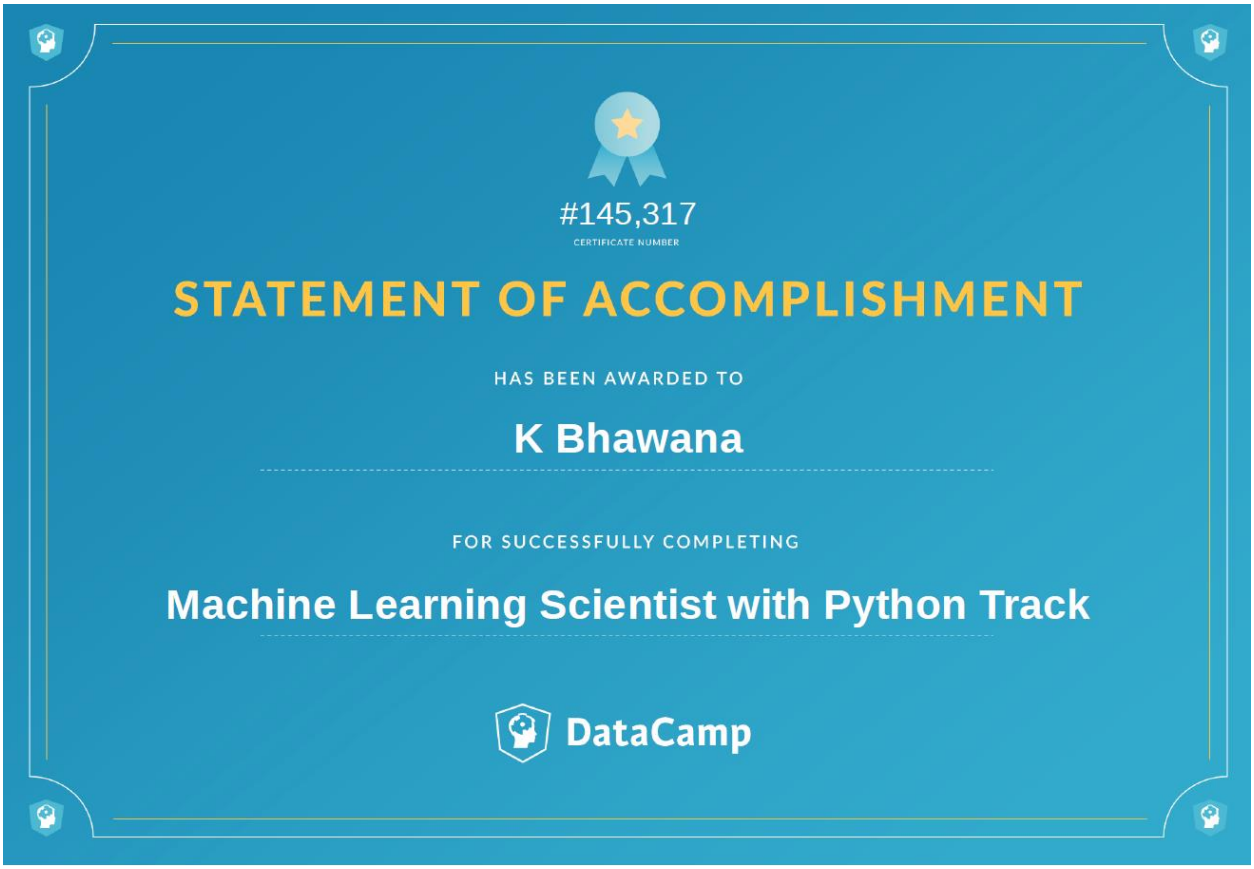

#145,317
CERTIFICATE NUMBER
STATEMENT OF ACCOMPLISHMENT
HAS BEEN AWARDED TO
K Bhawana
FOR SUCCESSFULLY COMPLETING
Machine Learning Scientist with Python Track
DataCamp

TABLE OF CONTENTS

I. INTRODUCTION OF THE PROJECT UNDERTAKEN

Risk management processes form an integral part of the insurance industry. Insurers consider every available quantifiable factor to develop profiles of high and low insurance risk for their prospective policyholders.

Level of risk determines the insurance premiums of these policies. To this end, insurers collect a vast amount of information about policyholders and insured objects. Increasingly, statistical methods and tools based on data mining techniques are being used to determine insurance policy risk levels.

We discuss the use of predictive analytics for assessing insurance risk in the following sections.

1. Data Pre-Processing
2. Exploratory Data Analysis
3. Feature Transformation
4. Model Creation
5. Ensemble Method

Before we start with predictive analysis of our insurance dataset, lets define more about machine learning and its various algorithms like KNN, SVM, Logistic Regression, Tree Base Models and their loss functions.

Machine Learning

Machine Learning is a subfield of Artificial Intelligence (AI), a topic of computer science born in the 1950s with the goal of trying to get computers to think or provide a level of automated intelligence similar to that of us humans.

Early success in AI was achieved by using an extensive set of defined rules, known as symbolic AI, allowing expert decision making to be mimicked by computers. This approach worked well for many domains but had a big shortfall in that in order to create an expert, you needed one.

Not only this, but also their expertise needed to be digitized somehow, which normally required explicit programming.

ML provides an alternative; instead of having to handcraft rules, it learns from examples and experience. It also differs from classical programming in that it is probabilistic as opposed to being discrete. That is, it is able to handle fuzziness or uncertainty much better than its counterpart, which will likely fail when given an ambiguous input that wasn't explicitly identified and handled.

Further it is divided into 3 parts

1. Supervised Learning
2. Unsupervised Learning
3. Semi-Supervised Learning

Supervised Learning

It is branch of machine learning where we have to train our model in supervised manner. We have all the data given to us, both dependent variable and independent variable. In supervised learning the algorithm try to learn the patterns of data with the help of underlying algorithms such as KNN, Logistic regression, Linear regression etc. All these algorithms are used in supervised learning. We are going to use supervised learning in this following project.

Regression vs Classification in Machine Learning

Regression Models

Regression Models anticipate a continuous value dependent on the information factors. The fundamental objective of regression issues is to assess a planning capacity dependent on the information and yield factors. On the off chance that your objective variable is an amount like pay, scores, tallness or weight, or the likelihood of a paired class (like the likelihood of downpour specifically areas), at that point you should utilize the relapse model.

Classification Models

Classification is a predictive model that approximates a planning capacity from input factors to distinguish discrete yield factors, that can be names or classifications. The planning capacity of order calculations is answerable for foreseeing the name or classification of the given information factors. A classification calculation can have both discrete and genuine esteemed factors, however it necessitates that the models be characterized into one of at least two classes. In this project we will use different type of classification models explained below:

1. Logistic Regression

Logistic regression is used to predict a discrete outcome based on variables which may be discrete, continuous or mixed. Thus, when the dependent variable has two or more discrete outcomes, logistic regression is a commonly used technique. The outcome could be in the form of Yes / No, 1 / 0, True / False, High/Low, given a set of independent variables. It is a parametric model in nature which means it assumes that data points are linearly separable. Logistic regression tries to minimize logistic loss. It can handle large number of features with good time space complexity.

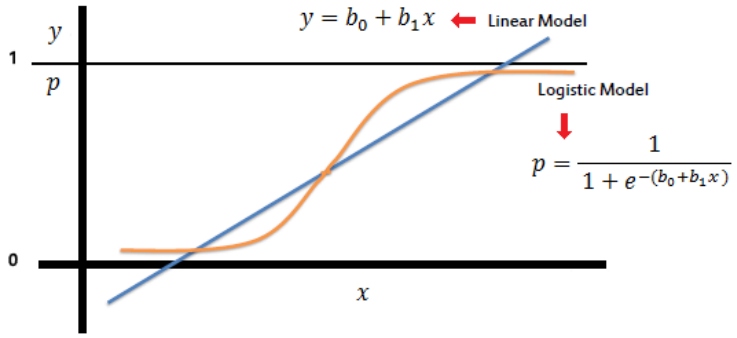


Fig 1.1

2. KNN

K Nearest Neighbour is the most simplest model in machine learning. It is non parametric in nature which means it doesn't assume anything about data unlike Logistic Regression. KNN separates the classes with the help of nearest neighbour of data points, it measure distance between datapoints and make a cluster of nearest data points. Distance can be Euclidean distance, Manhattan Distance or Hamming distance. This model does not generalize well on data hence also known as lazy learner.

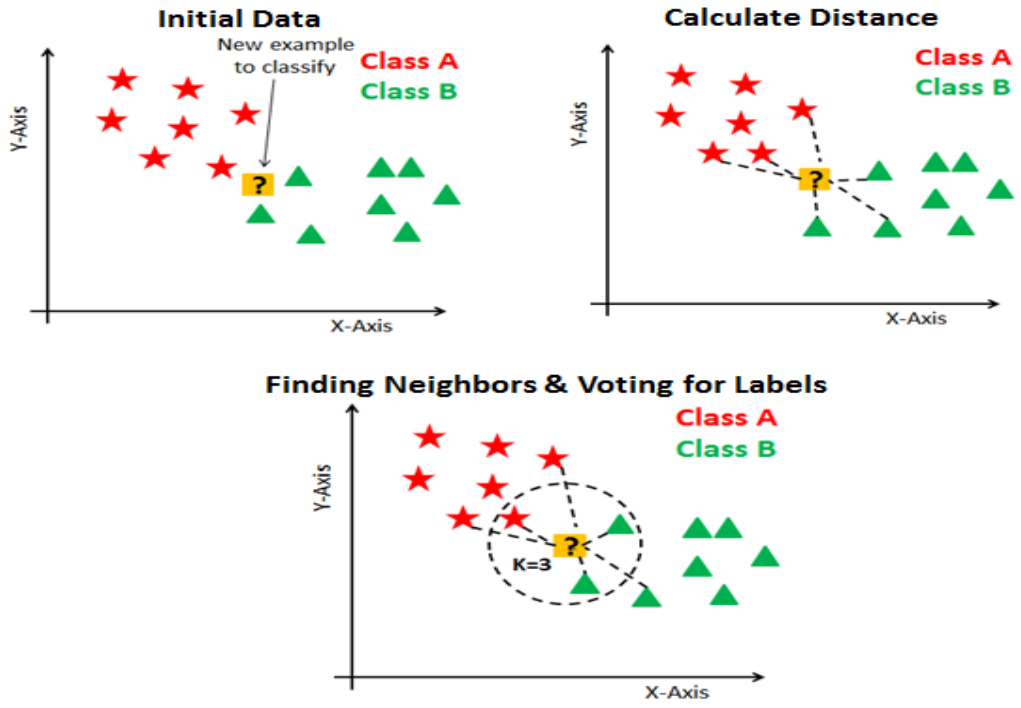


Fig 1.2

3. Decision Tree

Like SVMs, Decision Trees are versatile Machine Learning algorithms that can perform both classification and regression tasks, and even multioutput tasks. They are powerful algorithms, capable of fitting complex datasets.

Decision Trees are also the fundamental components of Random Forests ,which are among the most powerful Machine Learning algorithms available today.

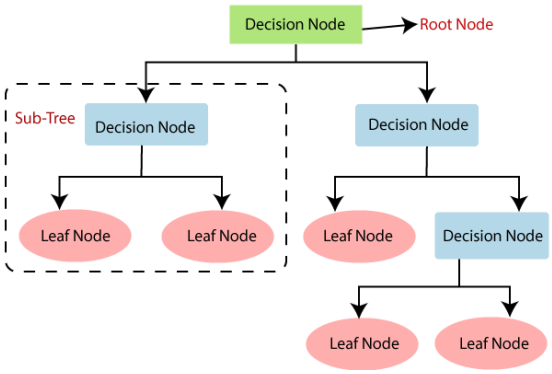


Fig 1.3

4. Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.Random decision forests correct for decision trees' habit of overfitting to their training set. Random forests generally outperform decision trees, but their accuracy is lower than gradient boosted trees. However, data characteristics can affect their performance.

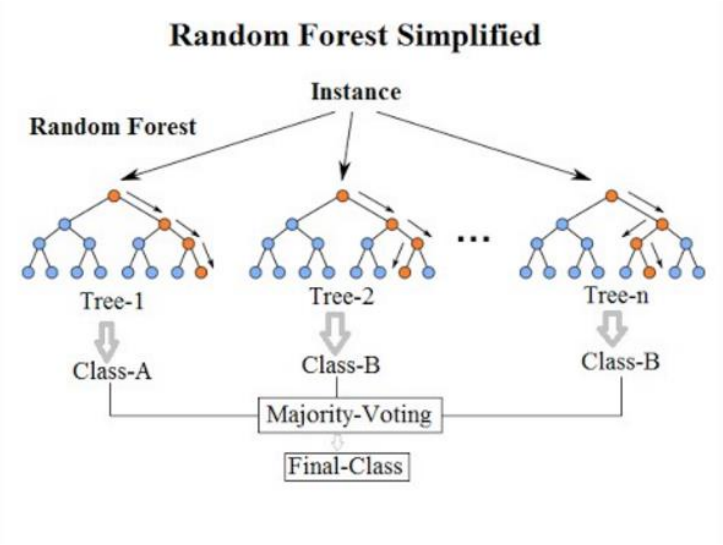


Fig 1.4

5. GBDT

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

4. Ensemble Models

If we aggregate the predictions of a group of predictors (such as classifiers or regressors), we will often get better predictions than with the best individual predictor. A group of predictors is called an ensemble; thus, this technique is called Ensemble Learning, and an Ensemble Learning algorithm is called an Ensemble method.

As an example of an Ensemble method, we can train a group of Decision Tree classifiers, each on a different random subset of the training set. To make predictions, you obtain the predictions of all the individual trees, then predict the class that gets the most votes. Such an ensemble of Decision Trees is called a Random Forest, and despite its simplicity, this is one of the most powerful Machine Learning algorithms available today. There is another powerful ensemble model known as XGBoost models which is also based on tree based models. It is one of the most robust models in machine learning.

Importance and Scope

Predictive models analytics and risk analytics enable underwriters to get an automated result to guide them in the decision-making stage. Given the dynamic nature of data collected, it is important to employ predictive modelling to get consistent and automated results. In doing so, insurance companies can also reduce costs, improve the experience of their clients, and generate substantial business growth.

Since predictive modelling and risk analytics can reveal hidden patterns in data, it gives a clearer insight of the impending risks. This in turn helps organizations to take preventive measures as well as adds to their learning experience. In the long run, organizations can strategize a new methodology to avoid risks, given their rich experience with big data analytics.

Predictive analytics can be used by underwriting to upsell and cross-sell as well – when the risk assessment is a little more automated they can focus on customization and selling of products as well.

Predictive analytics Applications

Organizations today use predictive analytics in a virtually endless number of ways. The technology helps adopters in fields as diverse as finance, healthcare, retailing, hospitality, pharmaceuticals, automotive, aerospace and manufacturing. Here are a few examples :

- Aerospace: Predict the impact of specific maintenance operations on aircraft reliability, fuel use, availability and uptime.
- Automotive: Incorporate records of component sturdiness and failure into upcoming vehicle manufacturing plans. Study driver behavior to develop better driver assistance technologies and, eventually, autonomous vehicles.
- Energy: Forecast long-term price and demand ratios. Determine the impact of weather events, equipment failure, regulations and other variables on service costs.
- Financial services: Develop credit risk models. Forecast financial market trends. Predict the impact of new policies, laws and regulations on businesses and markets.
- Manufacturing: Predict the location and rate of machine failures. Optimize raw material deliveries based on projected future demands.
- Law enforcement: Use crime trend data to define neighborhoods that may need additional protection at certain times of the year.
- Retail: Follow an online customer in real-time to determine whether providing additional product information or incentives will increase the likelihood of a completed transaction.

II. Exploratory data analysis

The data consists of records of roughly 52000 instances and 11 features.

There are 10 independent variables and 1 target that describes whether the insurances will be claimed or not.

- Target: Claim Status (Claim)
- Name of agency (Agency)
- Type of travel insurance agencies (Agency.Type)
- Distribution channel of travel insurance agencies (Distribution.Channel)
- Name of the travel insurance products (Product.Name)
- Duration of travel (Duration)
- Destination of travel (Destination)
- Amount of sales of travel insurance policies (Net.Sales)
- The commission received for travel insurance agency (Commission)
- Age of insured (Age)
- The identification record of every observation (ID)

Claim Status is given by 0 or 1, 0 means the claim is false and 1 means the claim is genuine. Here is the distribution of target variable in our dataset. As we can see in the below image our data is highly imbalance.

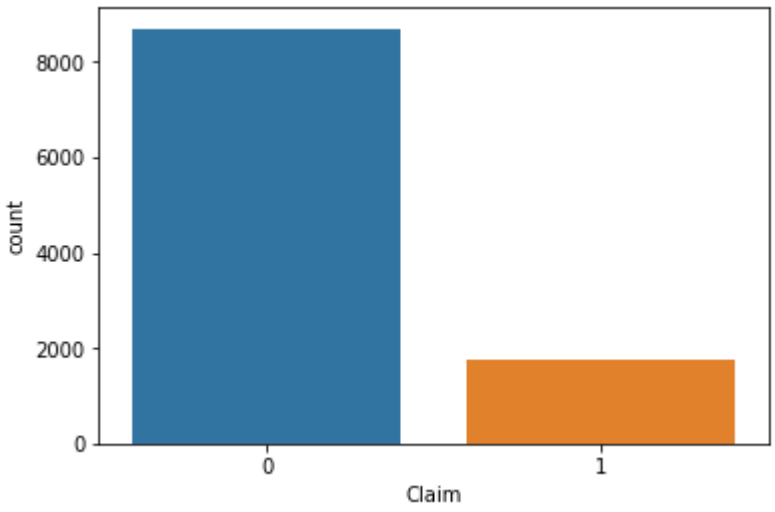


Fig 2.1

Our Dataset contains both numerical and text data types , We need to convert and pre-process the data into suitable format so that our predictive models can run properly. Before that lets do some basic analysis of our numerical data :

Univariate analysis of numerical Data

1. Age

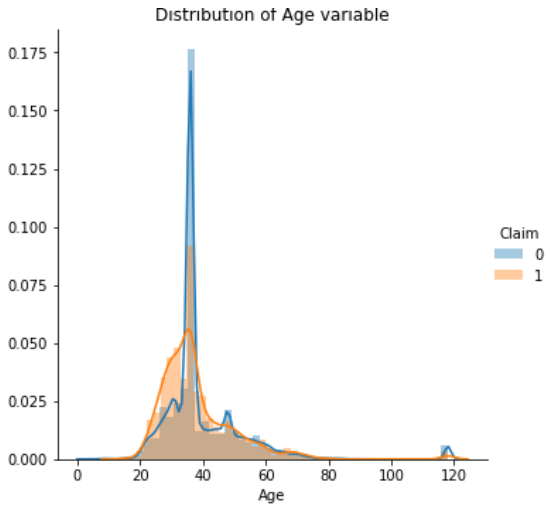


Fig 2.2

2 . Commision (in value)

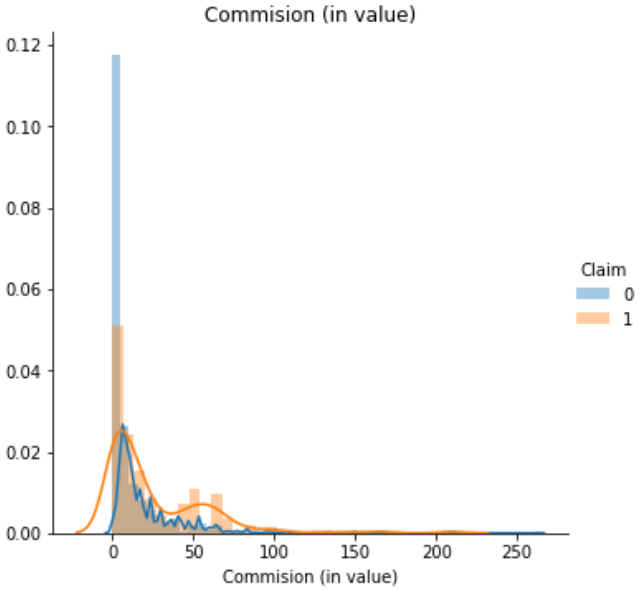


Fig 2.3

3. Duration

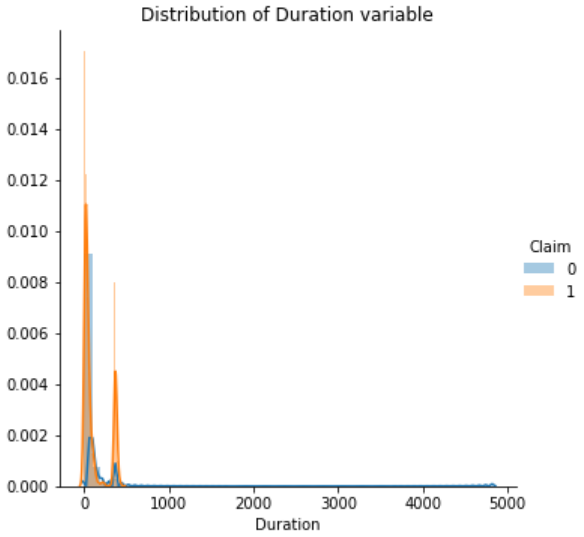


Fig 2.4

From the above distribution plot we can see that data is not linearly seperated so we will use complex models like ensemble and decision tree which does not assume the distribution of data. Lets see the distribution of label data points in both train and test split

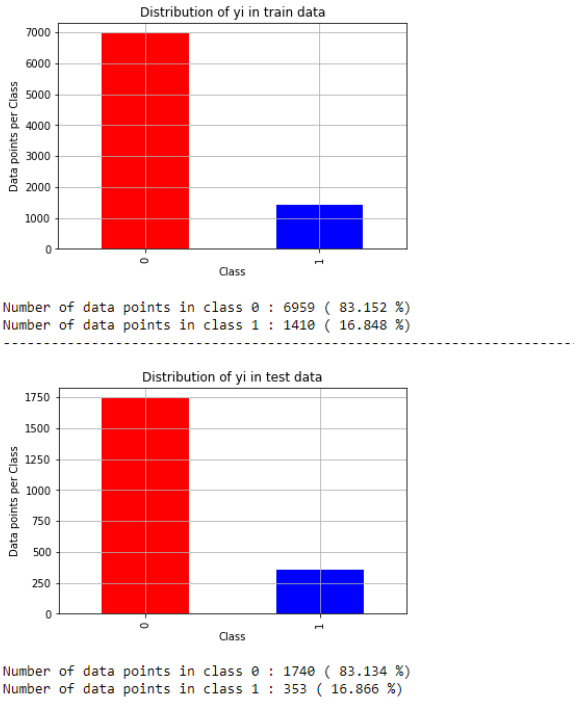


Fig 2.5

III. Feature Transformation

We need to transform our features into suitable format so that our predective model can work with out any undesrirable error. We have 2 types of data types, numerical and text data, Lets work with numerical data first. This is basic statistics of our numerical data

	Duration	Net Sales	Commision (in value)	Age	Claim
count	10462.000000	10462.000000	10462.000000	10462.000000	10462.000000
mean	59.376219	48.900969	12.373642	39.485089	0.168515
std	120.266449	61.848927	23.478199	13.485511	0.374341
min	0.000000	-357.500000	0.000000	3.000000	0.000000
25%	10.000000	20.000000	0.000000	34.000000	0.000000
50%	24.000000	29.000000	0.000000	36.000000	0.000000
75%	59.000000	54.500000	13.380000	43.000000	0.000000
max	4844.000000	666.000000	262.600000	118.000000	1.000000

Fig 3.2

As we can observe here, minimum and maximum values of some features are very low and high, also standard deviation of our data is not centered around mean. This can be because of outliers or some human error while entering the data. We need to standardize our data in order to make all the data points to center arount mean. We can do this with simple standard scale function of sklearn. After standardization in below image we can observe that most of our data is centered around mean because the standard deviation is 1 and mean is close to 0. Our numerical data is ready for modeling. Lets work with text data now.

	Duration	Net Sales	Commision (in value)	Age
count	8369.000000	8369.000000	8369.000000	8369.000000
mean	0.509688	0.788140	0.520107	2.961104
std	1.000060	1.000060	1.000060	1.000060
min	0.000000	-5.801536	0.000000	0.225808
25%	0.086887	0.324561	0.000000	2.559160
50%	0.208529	0.478728	0.000000	2.709699
75%	0.503945	0.876316	0.559423	3.236585
max	42.088104	10.807897	11.084967	8.881792

Fig 3.2

These features in below image are the text data.

	Agency	Agency Type	Distribution Channel	Product Name	Destination
27324	EPX	Travel Agency	Online	2 way Comprehensive Plan	UNITED STATES
24166	EPX	Travel Agency	Online	1 way Comprehensive Plan	UNITED KINGDOM
33902	C2B	Airlines	Online	Bronze Plan	SINGAPORE
10851	EPX	Travel Agency	Online	Cancellation Plan	MALAYSIA
33558	EPX	Travel Agency	Online	Cancellation Plan	PHILIPPINES
...	...	...	...	...	...
51863	C2B	Airlines	Online	Annual Silver Plan	SINGAPORE
20885	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	CROATIA
20474	EPX	Travel Agency	Online	Cancellation Plan	THAILAND
34082	EPX	Travel Agency	Online	Cancellation Plan	CHINA
50116	EPX	Travel Agency	Online	1 way Comprehensive Plan	SINGAPORE

Fig 3.3

Predective models can not handle text data so we need to convert it into some relevant numerical format. We will use one hot encoding technique which is available in sklearn library. In one hote encoding, the features are encoded into long sparse vectors where only 1 value is true and rest of them are 0. This will increase dimensionality of data but it will be efficient for our predictive modeling.

After converting text data into one hot encoded vectors and scaling the numerical data, we need to merge them togther for our model. This is the final data for our models. We have total now 126 features.

	0	1	2	3	4	5	6	7	8	9	...	116	117	118	119	120	121	Duration	Net Sales	Commision (in value)	Age
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	0.0	1.073313	0.545378	0.000000	2.325238
1	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	2.872690	4.127796	2.715343	3.750383
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	1.033853	0.163251	0.000000	2.100215
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.473520	0.310176	0.000000	2.700276
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.118380	-0.473427	0.000000	2.700276
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8364	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.276220	0.326502	0.000000	2.025207
8365	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.189408	0.261201	0.171830	2.775284
8366	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.473520	0.571378	0.000000	2.700276
8367	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.181516	0.473427	0.411103	5.175529
8368	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.220976	0.653003	0.601405	4.275437

8369 rows × 126 columns

Fig 3.4

IV. Models

We are using roc auc score for tuning of hyperparameters of models and varoius classification reports such as precision score , recall and roc

1. KNN

Lets start our predictive modeling with simplest model K nearest neighbour, We are tuning K neighbours using gridsearch with auc metric. We can see that at K =11 , our train and cv roc auc are high.

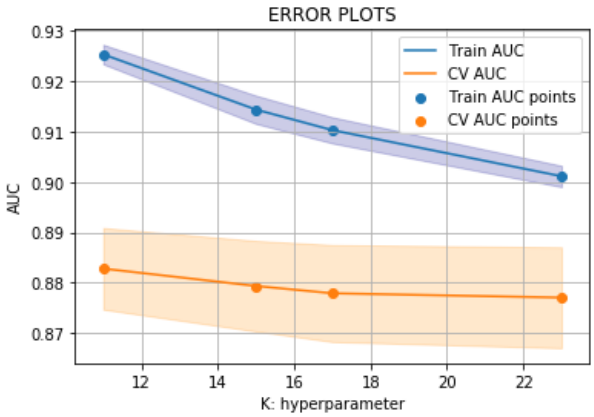


Fig 4.1

We will train our model on K = 11 hyper parameter

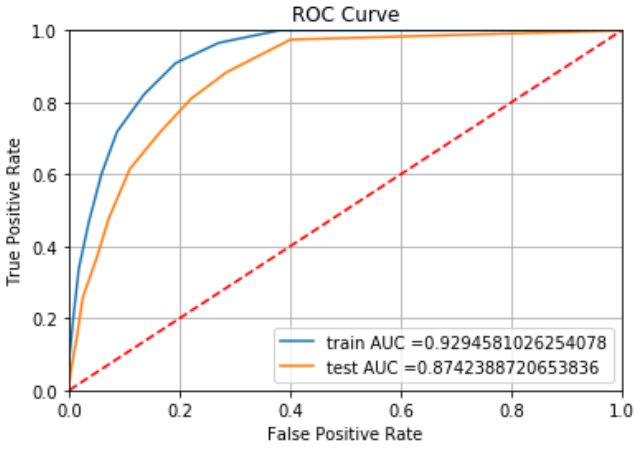


Fig 4.2

From this simple model we got the pretty good roc auc on test data is 87%

Confusion Matrix , Precision Matrix , Recall Matrix

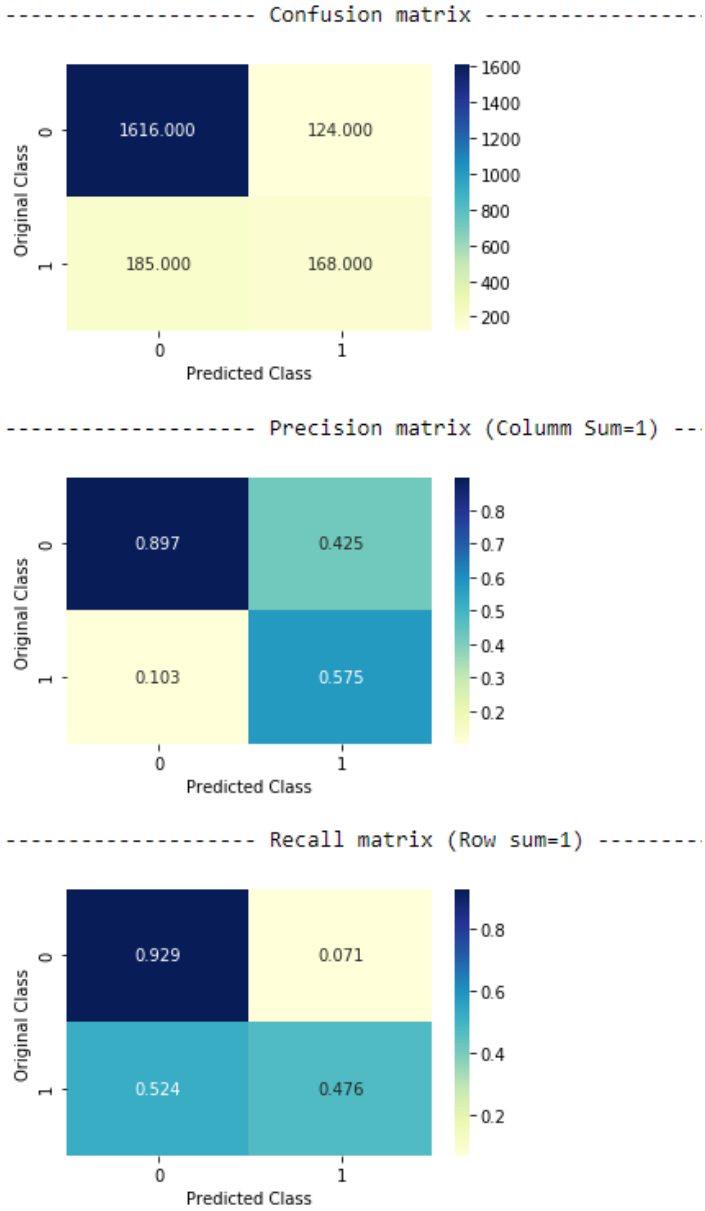
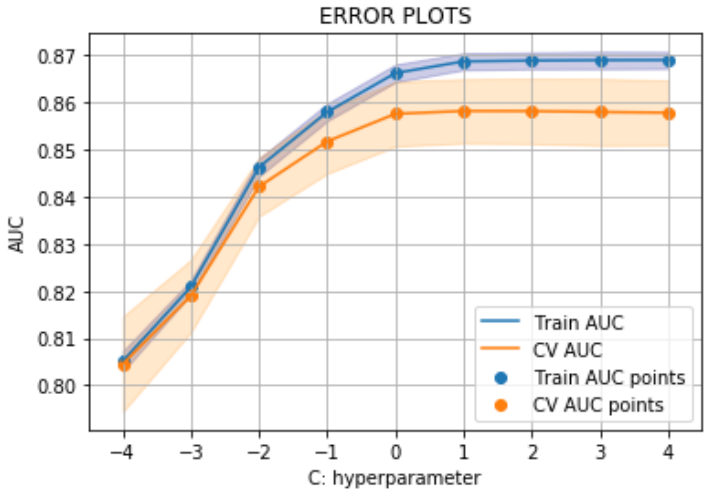


Fig 4.2

Despite of getting good accuracy our recall matrix is very bad. Lets see the improvement in logistic regression.

2. Logistic Regression

Lets use more powerfull model logistic regression. But since parametric nature of this model it assumes that data is linearly seperated and we saw in univariate analysis that data is not linearly seperated. Hence this model is working worse than KNN.




```
best c using L2 : 10
Wall time: 5.02 s
```

Fig 4.3

We will train our model on L2 = 10 hyper parameter

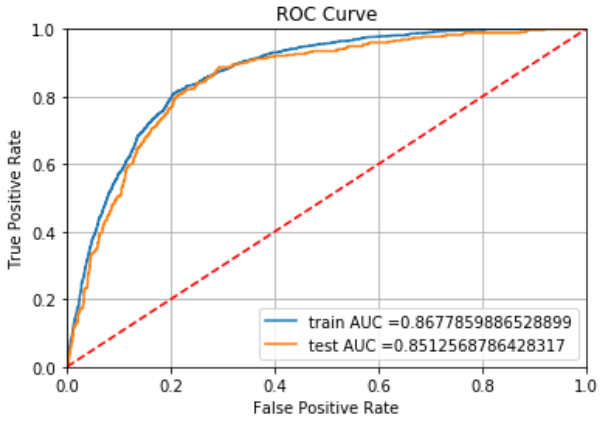


Fig 4.4

This is certainly very low roc auc than KNN but lets analyse the confusion matrix of this model

Confusion Matrix , Precision Matrix , Recall Matrix

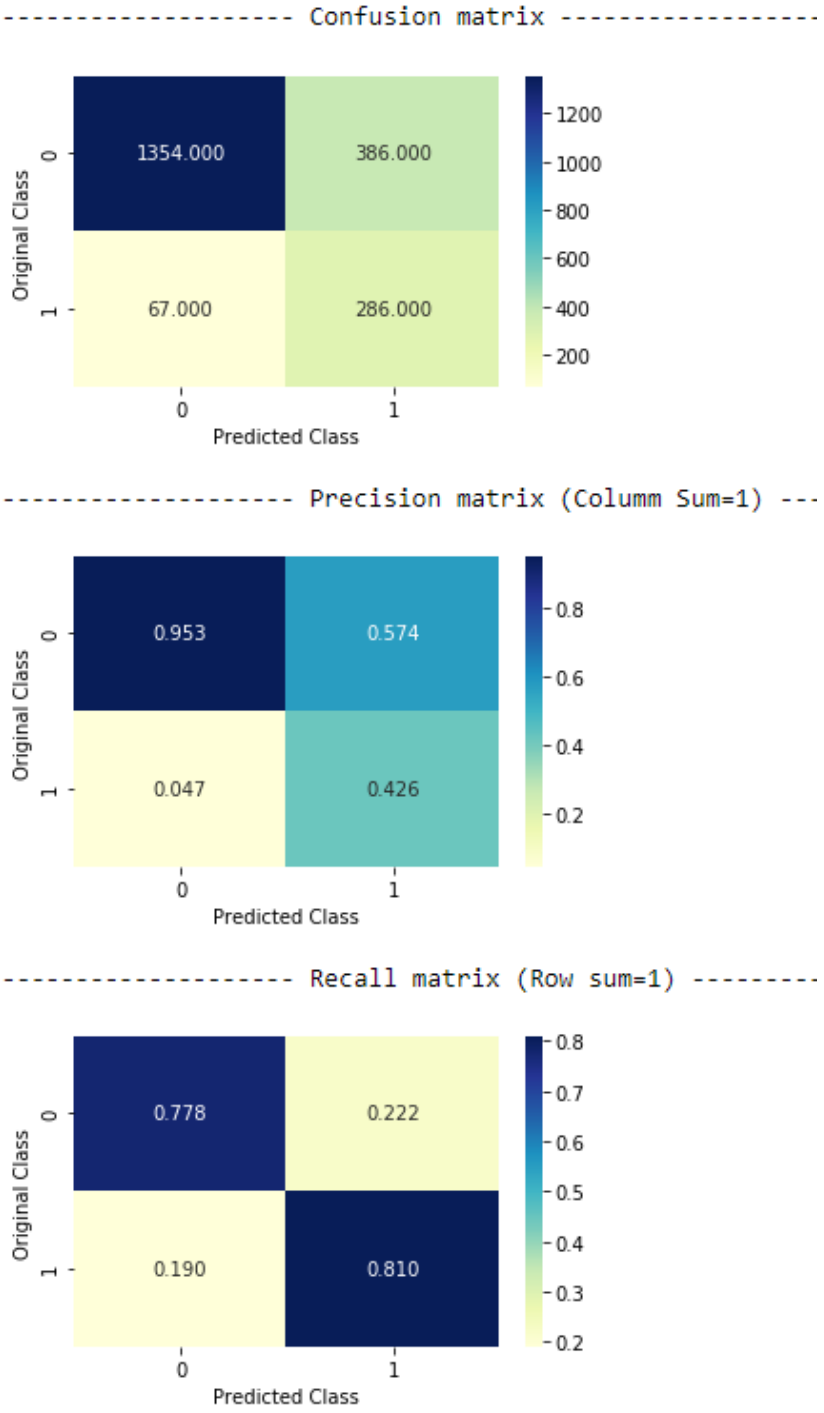
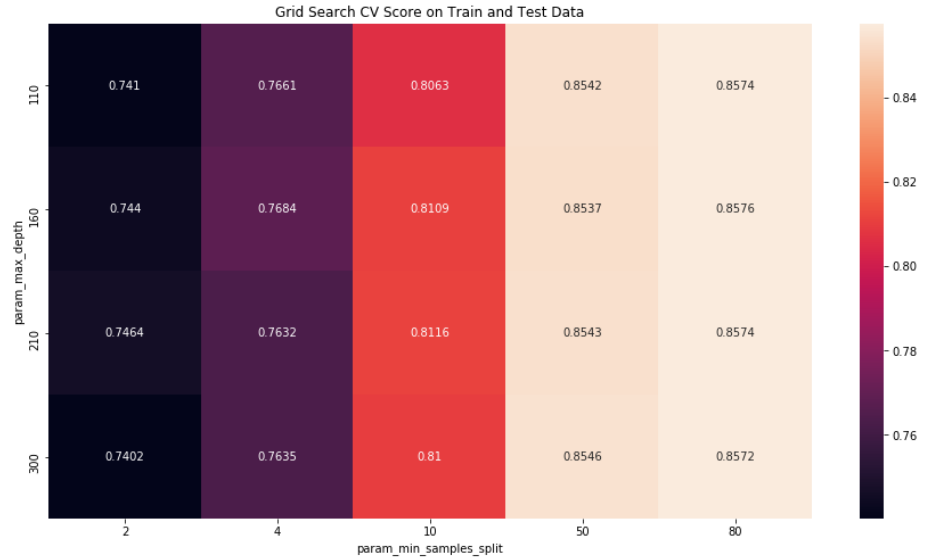


Fig 4.5

Recall is very much improved as compare to KNN model so we don't have to trust always in accuracy metric instead we need to analyse the confusion matrix.

3. Decision Tree

Since there are too many hyperparameters in this model we will only be tuning the 2 most important hyperparameter : max depth of tree and sample split at each node.




```
Best depth:  160
Best samples split:  80
Wall time: 2.38 s
```

Fig 4.6

We will train our model on above hyperparameters.

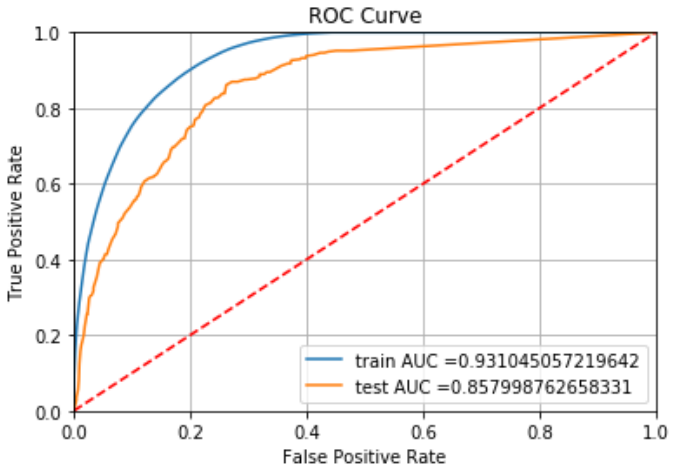


Fig 4.7

This model's roc auc is also below then our KNN but lets analyse the confusion matrix of Decision tree

Confusion Matrix , Precision Matrix , Recall Matrix

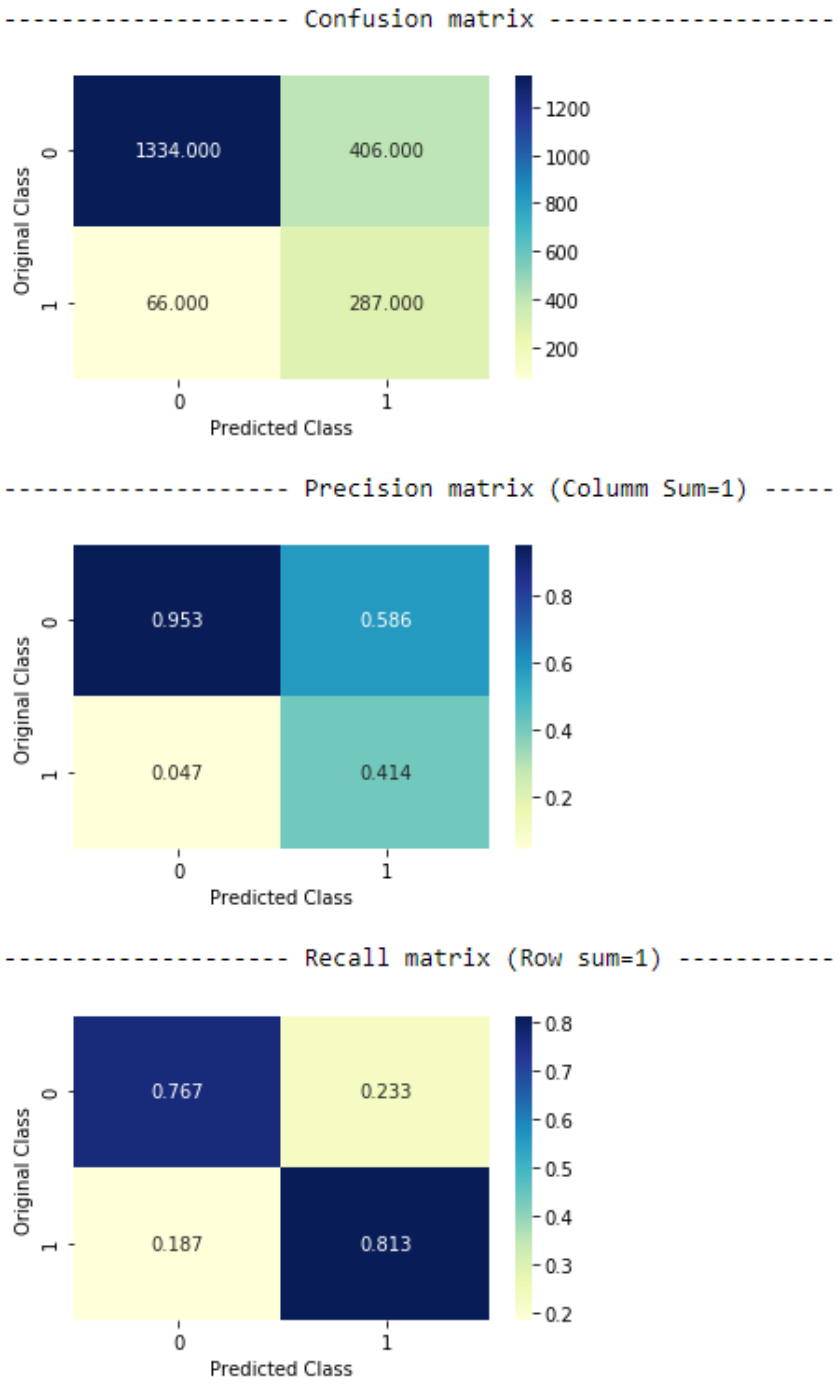


Fig 4.8

Recall is better than KNN model but still below than logistic regression model. Lets use more powerful algorithms such as ensemble models.

V. Ensemble Methods

1. Random Forest

Since there are too many hyperparameters in random forest we will only be tuning the 2 most important hyperparameter : max depth of tree and numbers of total trees.

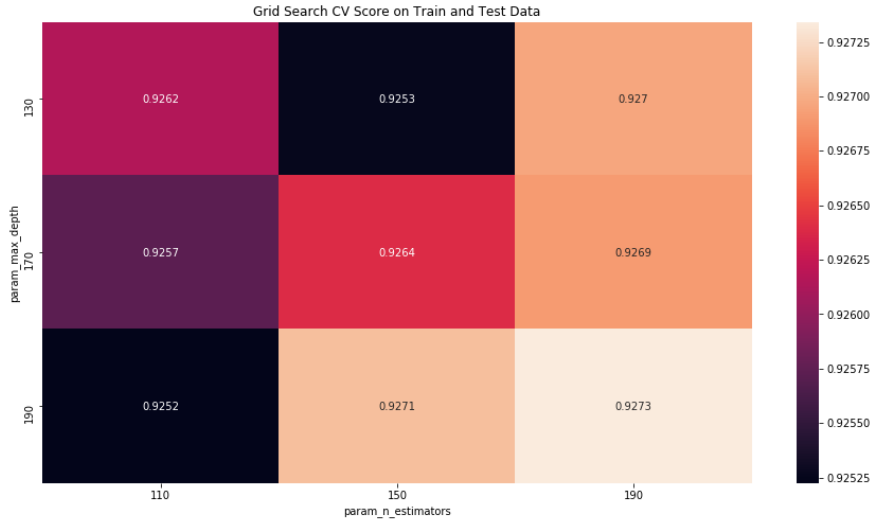


Fig 5.1

We will train our model on above hyperparameters.

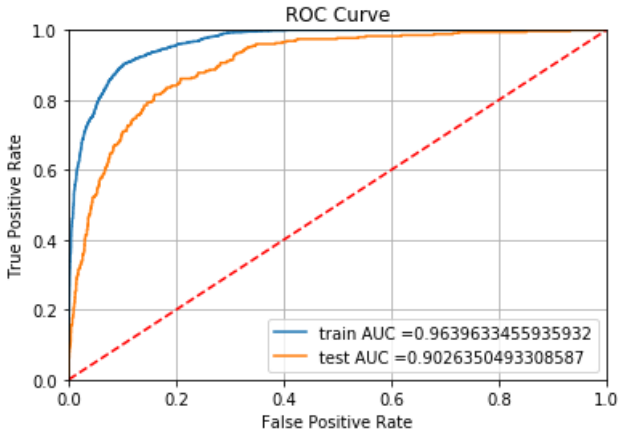


Fig 5.2

This is by far our best AUC ROC score. That's the power of ensemble methods. Lets analyse the confusion matrix also.

Confusion Matrix , Precision Matrix , Recall Matrix

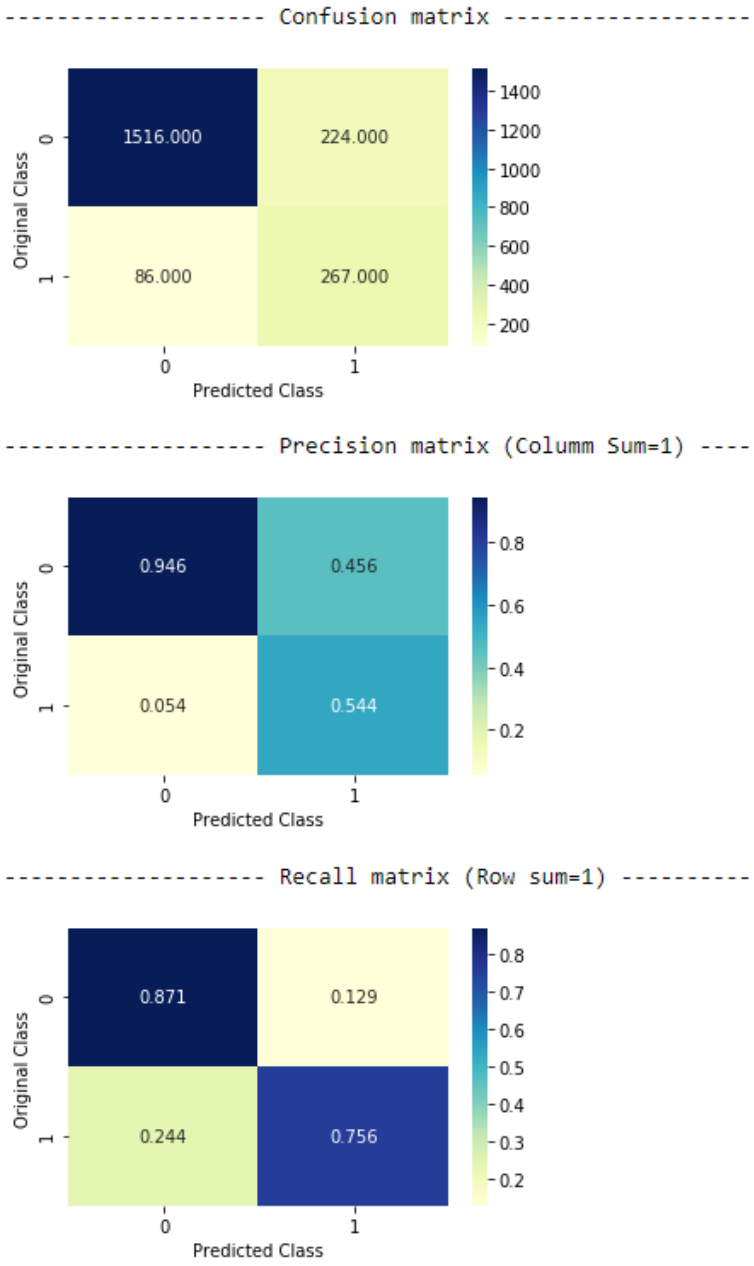


Fig 5.3

Our recall and precision matrix both have been improved darastically. Lets model another very powerful model GBDT.

2. GBDT using XGBOOST

We will use XGBOOST which is kind of GBDT but with good optimisation .Since there are too many hyperparameters in XGBOOST also we will only be tuning the 2 most important hyperparameter : max depth of tree and numbers of total trees.

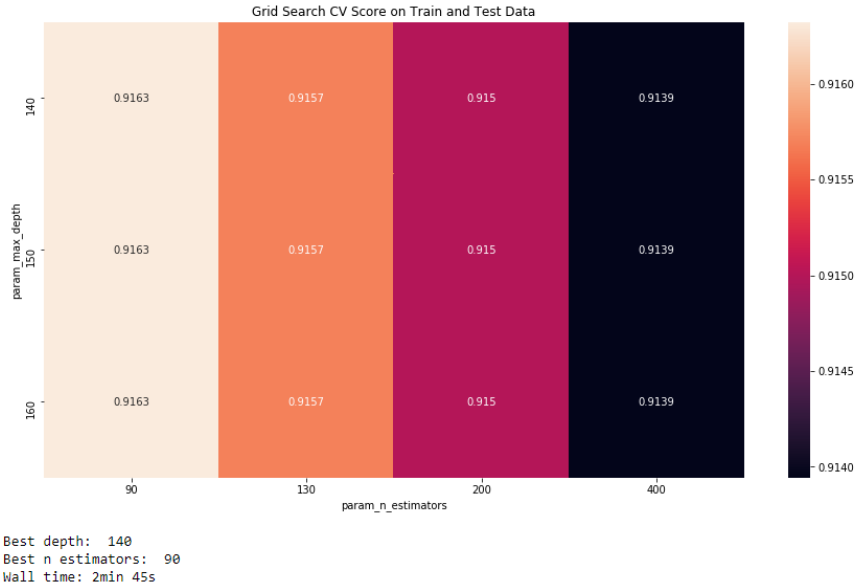


Fig 5.4

We will train our model on above hyperparameters.

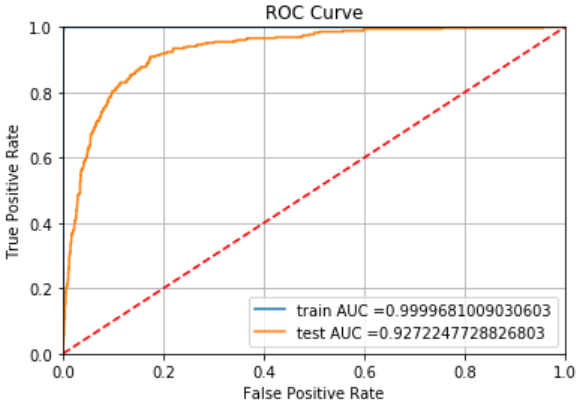


Fig 5.5

Look at the train roc auc of this model its almost equivalent to 100%. Xgboost can easilye be overfitted to data, but our test auc is also good so its over all it's a decent model and by far the best one of all the models lets analyse the confusion matrix.

Confusion Matrix , Precision Matrix , Recall Matrix

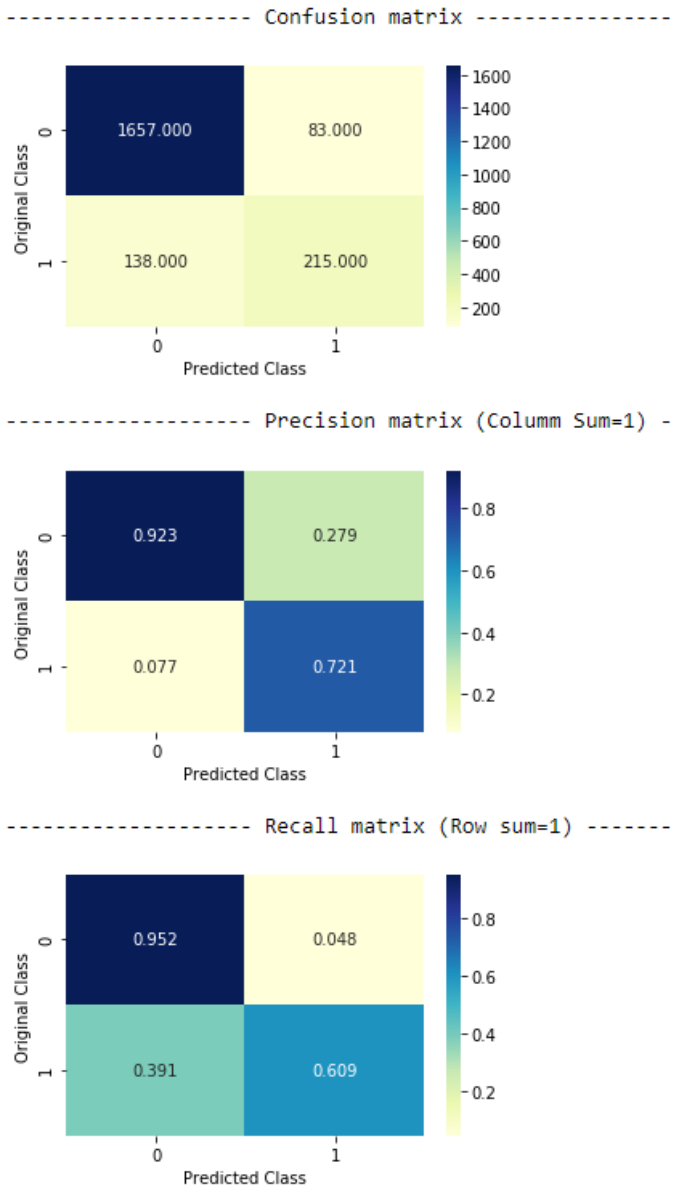


Fig 5.6

Class 0 has very high recall and precision because of the data imbalance. Our class 1 is not doing very good. Lets try to improve this with our final model.

3. Stacking

Here we will stack all the models we have used till now, named : KNN, Logistic Regression, Decision Tree, Random Forest, Xg Boost.

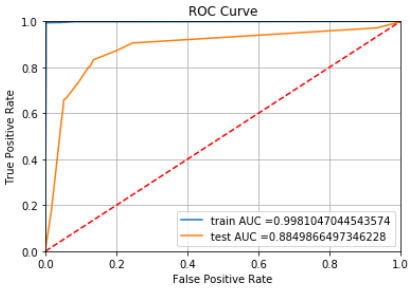


Fig 5.7

It has no hyperparameters since this is just stacking all the models.

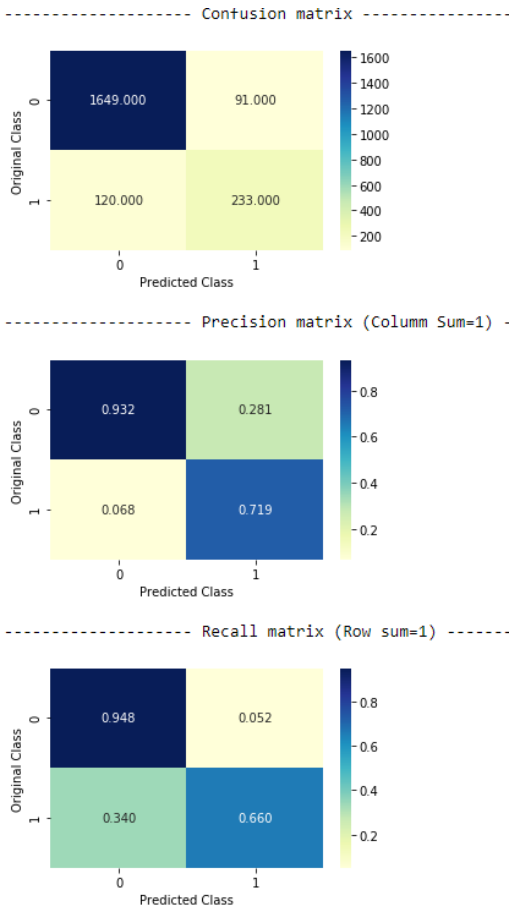


Fig 5.8

Our class 1 recall has improved in stacking while maintaing recall of class 0 as well.

VI. Code and all the imported packages

```
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import log_loss, confusion_matrix
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_curve, auc , precision_score , classification_report
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt

df = pd.read_csv('train.csv')
df = df.sample(frac=0.2, random_state=1)
df.head()

# # Spliting data into train and test

y_true = df.Claim.values
result = df
train_df, test_df,y_train, y_test = train_test_split(result, y_true, stratify = y_true, test_size = 0.2)
```

```
print(' Number of data points in train data :' ,train_df.shape[0])
print(' Number of data points in test data :' ,test_df.shape[0])
# # Exploratory data analysis
df.drop('ID',1).describe()
df.info()
ax = sns.countplot(x="Claim", data=df)
# ## Univariate Analysis
sns.FacetGrid(df, hue="Claim", size=5)    .map(sns.distplot, "Age")    .add_legend();
plt.title('Distribution of Age variable')
plt.show();
sns.FacetGrid(df, hue="Claim", size=5)    .map(sns.distplot, "Commision (in value)")
.add_legend();
plt.title('Commision (in value)')
plt.show();
sns.FacetGrid(df, hue="Claim", size=5)    .map(sns.distplot, "Duration")    .add_legend();
plt.title('Distribution of Duration variable')
plt.show();
# ### Distribution of y_i's in Train and test
train_class_distribution = train_df['Claim'].value_counts().sort_index()
test_class_distribution = test_df['Claim'].value_counts().sort_index()

train_class_distribution.plot(kind = 'bar', color=['r','b'])
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()
for i in range(0,2):
    print('Number of data points in class', i, ':',train_class_distribution.values[i], '(',
np.round((train_class_distribution.values[i]/train_df.shape[0]*100), 3), '%)')
print('-'*80)
```

```
test_class_distribution.plot(kind = 'bar', color=['r','b'])
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()
for i in range(0,2):
    print('Number of data points in class', i, ':',test_class_distribution.values[i], '(',
np.round((test_class_distribution.values[i]/test_df.shape[0]*100), 3), '%)')
# # Feature Transformation
#  OneHot Encoding
# # Standard Scaling of numerical columns

num_train = train_df.iloc[::,[5,7,8,9]]
num_test = test_df.iloc[::,[5,7,8,9]]
cols = list(num_train.columns)
cols

scalar = StandardScaler(with_mean=False)
scale_train = pd.DataFrame(scalar.fit_transform(num_train),columns=cols)
scale_test = pd.DataFrame(scalar.transform(num_test),columns=cols)

# ### After Standardization
scale_train.describe()

# # ONE HOT ENCODING
new_Xtrain = train_df.iloc[::,[1,2,3,4,6]]
new_Xtest = test_df.iloc[::,[1,2,3,4,6]]
cols = list(new_Xtrain.columns)
cols
```

```
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder(handle_unknown='ignore')
enc.fit(new_Xtrain)
new_Xtrain = enc.transform(new_Xtrain).toarray()
new_Xtest = enc.transform(new_Xtest).toarray()
new_Xtrain = pd.DataFrame(new_Xtrain)
new_Xtest = pd.DataFrame(new_Xtest)
# # Merging dataframes of categorical and numerical = OHE
final_train_ohe = pd.concat([new_Xtrain,scale_train],1)
final_test_ohe = pd.concat([new_Xtest,scale_test],1)
final_train_ohe
print('OHE X_train :', final_train_ohe.shape)
print('OHE X_test :', final_test_ohe.shape)
# # Machine Learning Models
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

    A =(((C.T)/(C.sum(axis=1))).T)

    B =(C/C.sum(axis=0))

    labels = [0,1]
    print("-"*20, "Confusion matrix", "-"*20)
    plt.figure(figsize=(5,3))
    sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels,
yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

```
    print("-"*20, "Precision matrix (Columm Sum=1)", "-"*20)
    plt.figure(figsize=(5,3))
    sns.heatmap(B, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels,
yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
    plt.figure(figsize=(5,3))
    sns.heatmap(A, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels,
yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

def knn_brute_force(Xtrain,ytrain):
    """
    Function to fine the best k neighbors.
    Input : Training data
    ---
    Output : best value of K
    """
    K = [11,15,17,23]#using odd numbers for K in knn
    neigh = KNeighborsClassifier(algorithm='brute')
    parameters = {'n_neighbors':K}
    clf = GridSearchCV(neigh, parameters, cv=5, scoring='roc_auc',return_train_score=True)
    clf.fit(Xtrain, ytrain)

    train_auc= clf.cv_results_['mean_train_score']
    train_auc_std= clf.cv_results_['std_train_score']
```

```
    cv_auc = clf.cv_results_['mean_test_score']
    cv_auc_std= clf.cv_results_['std_test_score']
    best_k= clf.best_params_
    best_k = list(best_k.values())[0]

    plt.plot(K, train_auc, label='Train AUC')
    plt.gca().fill_between(K,train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.2,color='darkblue')

    plt.plot(K, cv_auc, label='CV AUC')
    plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc +
cv_auc_std,alpha=0.2,color='darkorange')
    plt.scatter(K, train_auc, label='Train AUC points')
    plt.scatter(K, cv_auc, label='CV AUC points')
    plt.grid(True)
    plt.legend()
    plt.xlabel("K: hyperparameter")
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
    plt.show()
    return best_k

def final_knn_brute(Xtrain,ytrain,Xtest,ytest,best_K):
    """
    Function to find the model with best hyperparameter
    Input : Training data, Test Data and best value of k
    ---
    Output : best trained model
    """

    neigh = KNeighborsClassifier(n_neighbors=best_K, algorithm='brute')
```

```
    neigh.fit(Xtrain, ytrain)
    train_fpr, train_tpr, thresholds = roc_curve(ytrain, neigh.predict_proba(Xtrain)[:,1])
    test_fpr, test_tpr, thresholds = roc_curve(ytest, neigh.predict_proba(Xtest)[:,1])
    plt.grid(True)
    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
    plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
    plt.legend()
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curve")
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.show()
    return neigh


# # 1 .K Nearest Neighbour Classification with OHE

get_ipython().run_cell_magic('time', '', "best_K_ohe =
knn_brute_force(final_train_ohe,y_train)\nprint('Best K is :', best_K_ohe)")
knn_brutemodel_ohe = final_knn_brute(final_train_ohe,y_train,final_test_ohe,
y_test,best_K_ohe)
plot_confusion_matrix(y_test,knn_brutemodel_ohe.predict(final_test_ohe))


# # 2 . Logistic Regression
```

```
def logit(Xtrain,ytrain):
    """
    Function to find the best hyperparameter
    Input : Training Dataset
    ---
    Output : best hyperparameter
    """
    parameters = [{'C': [10**x for x in range(-4,5)]}]
    K =[10**x for x in range(-4,5)]
    K = np.log10(K)

    log= LogisticRegression(class_weight = 'balanced')
    clf = GridSearchCV(log, parameters, cv=5, scoring='roc_auc',return_train_score=True)
    clf.fit(Xtrain, ytrain)

    train_auc= clf.cv_results_['mean_train_score']
    train_auc_std= clf.cv_results_['std_train_score']
    cv_auc = clf.cv_results_['mean_test_score']
    cv_auc_std= clf.cv_results_['std_test_score']
    lamb= clf.best_params_
    lamb = list(lamb.values())[0]

    plt.plot(K, train_auc, label='Train AUC')
    # this code is copied from here: https://stackoverflow.com/a/48803361/4084039
    plt.gca().fill_between(K,train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.2,color='darkblue')

    plt.plot(K, cv_auc, label='CV AUC')
    plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc +
cv_auc_std,alpha=0.2,color='darkorange')
    plt.scatter(K, train_auc, label='Train AUC points')
```

```
    plt.scatter(K, cv_auc, label='CV AUC points')
    plt.grid(True)
    plt.legend()
    plt.xlabel("K: hyperparameter")
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
    plt.show()
    return lamb

def final_logl2(Xtrain,ytrain,Xtest,ytest,best_c):
    """
    Function to find the model with best hyperparameter
    Input : Training data, Test Data and best value of c
    ---
    Output : best trained model
    """
    logl2 = LogisticRegression(C= best_c,class_weight = 'balanced')
    logl2.fit(Xtrain, ytrain)
    train_fpr, train_tpr, thresholds = roc_curve(ytrain, logl2.predict_proba(Xtrain)[:,1])
    test_fpr, test_tpr, thresholds = roc_curve(ytest, logl2.predict_proba(Xtest)[:,1])
    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
    plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
    plt.legend()
    plt.grid(True)
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curve")
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.show()
```

```
    return logl2


# ###  Logistic Regression with OHE
get_ipython().run_cell_magic('time', '', "best_c_ohe = logit(final_train_ohe,y_train)\nprint('best
c using L2 :',best_c_ohe)")
log_model_ohe= final_logl2(final_train_ohe,y_train,final_test_ohe, y_test,best_c_ohe)
plot_confusion_matrix(y_test,log_model_ohe.predict(final_test_ohe))
# # 3 .Decision Tree
def dtc(Xtrain,ytrain):
    """
    Function to find the best hyperparameter
    Returns : best parameters of DecisionTree Calssifier
    ---
    Input : Dataset
    """
    max_depth = [110,160,210,300]
    min_samples_split = [2,4,10,50,80]
    parameters = {'max_depth': max_depth, 'min_samples_split' :min_samples_split}
    clftree = DecisionTreeClassifier()
    clf = GridSearchCV(clftree,parameters,cv=5, scoring='roc_auc',return_train_score=True)
    clf.fit(Xtrain,ytrain)
    tab = pd.DataFrame(clf.cv_results_)
    import seaborn as sns
    plt.figure(figsize=(15, 8))
    max_scores = tab.groupby(['param_max_depth','param_min_samples_split']).max()
    max_scores = max_scores.unstack()[['mean_test_score', 'mean_train_score']]
    sns.heatmap(max_scores.mean_test_score, annot=True, fmt='.4g')
    plt.title('Grid Search CV Score on Train and Test Data')
    plt.show()
    best_parameter= clf.best_params_
```

```
    print('Best depth: ', clf.best_estimator_.max_depth)
    print('Best samples split: ', clf.best_estimator_.min_samples_split)
    return clf.best_estimator_.max_depth, clf.best_estimator_.min_samples_split

def final_tree( Xtrain ,ytrain ,Xtest ,ytest , param1 , param2):
    """
    Returns : Model
    ---
    Input : Train dataset, Test Dataset , best parameters
    """
    from sklearn.metrics import roc_curve, auc
    import matplotlib.pyplot as plt
    from sklearn.tree import DecisionTreeClassifier

    clftre = DecisionTreeClassifier(class_weight='balanced',min_samples_split=param1,
max_depth=param2)
    clftre.fit(Xtrain,ytrain)
    train_fpr, train_tpr, thresholds = roc_curve(y_train, clftre.predict_proba(Xtrain)[:,1])
    test_fpr, test_tpr, thresholds = roc_curve(y_test, clftre.predict_proba(Xtest)[:,1])

    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
    plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
    plt.legend()
    plt.grid(True)
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curve")
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.show()
```

```
    return  clftre

# # Decision Tree with OHE
get_ipython().run_cell_magic('time', '', 'best_depth_dt_ohe, best_samle_split_dt_ohe =
dtc(final_train_res,y_train)')
dt_ohe= final_tree(final_train_ohe,y_train, final_test_ohe,
y_test,best_samle_split_dt_ohe,best_depth_dt_ohe)
plot_confusion_matrix(y_test,dt_ohe.predict(final_test_ohe))



# # 4 Random Forest
def best_RF(Xtrain,ytrain):
    """
    Function to find the best hyperparameter of random forest
    Returns : best depth and best number of models
    ---
    Input : Training dataset
    """
    # https://scikit-
learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
    max_depth = [130,170,190,]
    n_models = [110,150,190]
    parameters = {'max_depth':  max_depth, 'n_estimators' :n_models}
    clftree = RandomForestClassifier(class_weight='balanced')
    clf = GridSearchCV(clftree,parameters,cv=5, scoring='roc_auc',return_train_score=True)
    clf.fit(Xtrain,ytrain)
    tab = pd.DataFrame(clf.cv_results_)
    import seaborn as sns
    plt.figure(figsize=(15, 8))
    max_scores = tab.groupby(['param_max_depth','param_n_estimators']).max()
    max_scores = max_scores.unstack()[['mean_test_score', 'mean_train_score']]
```

```
    sns.heatmap(max_scores.mean_test_score, annot=True, fmt='.4g')
    plt.title('Grid Search CV Score on Train and Test Data')
    plt.show()
    best_parameter= clf.best_params_
    print('Best depth: ', clf.best_estimator_.max_depth)
    print('Best n estimators: ', clf.best_estimator_.n_estimators)
    return clf.best_estimator_.max_depth, clf.best_estimator_.n_estimators

def final_RF( Xtrain ,ytrain ,Xtest ,ytest , param1 , param2):
    """
    Returns : threshold values, False positive rate , True positive rate and the trained model
    ---
    Input : Train dataset, Test Dataset , best parameters
    """
    clftre = RandomForestClassifier(class_weight='balanced',n_estimators=param1,
max_depth=param2,criterion='entropy',min_samples_leaf=3)
    clftre.fit(Xtrain,ytrain)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the
positive class
    # not the predicted outputs
    train_fpr, train_tpr, thresholds = roc_curve(ytrain, clftre.predict_proba(Xtrain)[:,1])
    test_fpr, test_tpr, thresholds = roc_curve(ytest, clftre.predict_proba(Xtest)[:,1])

    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
    plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
    plt.legend()
    plt.grid(True)
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curve")
    plt.plot([0, 1], [0, 1],'r--')
```

```
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.show()
    return  clftre


# # Random Forest with OHE
get_ipython().run_cell_magic('time', '', 'best_depth_rf_ohe, best_samle_split_rf_ohe =
best_RF(final_train_ohe,y_train)')
rf_ohe= final_RF(final_train_ohe,y_train, final_test_ohe, y_test,200,250)
plot_confusion_matrix(y_test,rf_ohe.predict(final_test_ohe))
# # 5. GBDT using Xgboost
def best_GBDT(Xtrain,ytrain):
    """
    Input : Training Data
    ---
    Output :Best parameters
    """
    max_depth = [140,150]
    n_models = [130,200]
    parameters = {'max_depth': max_depth, 'n_estimators' :n_models}
    clftree = XGBClassifier(booster='gbtree')
    clf = GridSearchCV(clftree,parameters,cv=3, scoring='roc_auc',return_train_score=True)
    clf.fit(Xtrain,y_train)
    tab = pd.DataFrame(clf.cv_results_)
    import seaborn as sns
    plt.figure(figsize=(15, 8))
    max_scores = tab.groupby(['param_max_depth','param_n_estimators']).max()
    max_scores = max_scores.unstack()[['mean_test_score', 'mean_train_score']]
    sns.heatmap(max_scores.mean_test_score, annot=True, fmt='.4g')
    plt.title('Grid Search CV Score on Train and Test Data')
```

```
    plt.show()
    best_parameter= clf.best_params_
    print('Best depth: ', clf.best_estimator_.max_depth)
    print('Best n estimators: ', clf.best_estimator_.n_estimators)
    return clf.best_estimator_.max_depth, clf.best_estimator_.n_estimators

def final_GBDT( Xtrain ,ytrain ,Xtest ,ytest , param1 , param2):
    """
    Returns : threshold values, False positive rate , True positive rate and the trained model
    ---
    Input : Train dataset, Test Dataset , best parameters
    """
    from sklearn.metrics import roc_curve, auc
    import matplotlib.pyplot as plt
    from xgboost import XGBClassifier
    clftre = XGBClassifier(booster='gbtree', n_estimators=param1 , max_depth=param2)
    clftre.fit(Xtrain,ytrain)
    train_fpr, train_tpr, thresholds = roc_curve(ytrain, clftre.predict_proba(Xtrain)[:,1])
    test_fpr, test_tpr, thresholds = roc_curve(ytest, clftre.predict_proba(Xtest)[:,1])

    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
    plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
    plt.legend()
    plt.grid(True)
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curve")
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.show()
```

```
    return clftre


# #  GBDT with OHE
get_ipython().run_cell_magic('time', '', 'best_depth_xg_ohe, best_samle_split_xg_ohe =
best_GBDT(final_train_ohe,y_train)')
xg_ohe= final_GBDT(final_train_ohe,y_train, final_test_ohe,
y_test,best_samle_split_xg_ohe,best_depth_xg_ohe)
plot_confusion_matrix(y_test,xg_ohe.predict(final_test_ohe))
```

VII. Conclusion

Classification Models	Best Hyper-parameter	Train ROC AUC	Test ROC AUC	Precision	
				0	1
KNN (Brute Force)	K = 11	0.92	0.87	0.89	0.57
Logistic Regression(L2)	C=10	0.86	0.85	0.95	0.42
Decision Tree	Depth = 160 Sample Split = 80	0.93	0.85	0.95	0.41
Random Forest	Depth = 190 n_estimators = 190	0.96	0.90	0.94	0.54
GBDT(XGBoost)	Depth = 140 n_estimators = 90	0.99	0.92	0.92	0.72
Stacking Classifier	Using all models	0.99	0.88	0.93	0.71

Table 7.1

As we can see GBDT has the highest test accuracy and best precision for both classes as compare to other models.

Future Scope:

- ❑ XGBOOST is our best model but it was suffering in our class 1 datapoints
- ❑ We can improve our model with more data
- ❑ We can use feature selection and try to minimizing the overfitting of models

Applicability of the outcome of this project:

- To improve operational efficiency from claim registration to settlement.
- Enhancing the customer experience while reducing the claims settlement time.
- These insights can help a carrier save millions of dollars in claim costs through proactive management, fast settlement, targeted investigations and better case management.
- In doing so, insurance companies can also reduce costs and generate substantial business growth.

References:

- https://www.latentview.com/blog/insurance-risk-assessment-using-predictive-analytics/ (Accessed on 1st Sept 2020).

- https://learning.oreilly.com/library/view/hands-on-machine-learning/9781492032632/(Accessed on 12th Aug 2020).

- https://en.wikipedia.org/wiki/Logistic_regression#:~:text=Logistic%20regression%20is%20a%20statistical,a%20form%20of%20binary%20regression (Accessed on 13th Aug 2020)

- https://scikit-learn.org/stable/

- https://stackoverflow.com

- A. Géron, in Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (Ed.: 2nd) O'Reilly Media, Inc. 2019, pp. 3-1059.