REPORT

(FINAL ROUND COMPETITION)

WANDERERS (DS21-68)

AUC Score: 0.5995

Team members: Jathurshan Pradeepkumar (Leader)

Mithunjha Anandakumar

Vinith Kugathasan

Introduction

Problem Context

Life insurance is a contract between insurance company and policy owner, where the company pays an assured sum to the policy owner's family upon his/her death (most common use). Life insurance is a tool which help individuals to accomplish a variety of financial goals. There are many policies/products offered in a life insurance to a customer. E.g.: education policy, retirement policy, health policy and so on. A customer can buy either one product at a time and buy the add-ons later or he/she can buy multiple products at once and buy the add-ons later if required. Cross selling is the action of selling an additional product/service to an existing customer.

The task given is to identify active customers who are most likely to buy another life insurance policy in addition to existing ones (cross sell) within the next 6 months, also to predict what product will the customer most likely to buy. Cross-selling are one of the best and easiest methods of generating additional revenue for an insurance company. Cross selling not only generate more revenue but also may carry the advantage of strengthening the customer relationships.

Aims & Objectives

The Insurance company's focus is to increase their revenue, which will get improved by both upsells and cross sells. Thus, the model they should employ should focus on accurately detecting the cross-sell probability to enhance their business. Thus, the model they should employ should focus on accurately detecting the customer with higher chance of buying an additional product in addition to the existing one. This can be achieved with the aid of the lead/important features hyper tuned for higher recall. Such model will help to shortlist the customer with high cross-selling probability. Accurate prediction can help agent to identify the potential cross selling customer and advertise/ promote him to go for a more expensive or advanced product. Also, accurate prediction will help agents to make targeted proactive interventions as there is a vast potential to increase cross sell rate.

Data

Data Preprocessing

- The initial step was to extract target variables from the given dataset.
- Then the Nominal data fields in the string format such as Policy payment mode, policy status, main holder smoker flag, payment method, etc. were converted into binary variable using the one hot encoding.
- we checked the dataset for N/A values using (isna()) in the dataset, several were found. We impute handling was done to handle the not defined values. E.g.: for some clients, the age was not given in that case were imputed with the mean age for all undefined values.

Model Features

Features

model = RandomForestClassifier(max_depth = 15, n_estimators = 115, class_weight = 'balanced', random_state = 39)

Above state code gives the parameters used for hyper parameter tuning: Max_depth, n_estimators, class_weight, random_state

Features Engineering

In this dataset the target variables are not given explicitly. So, attempts to extract target variable were done. For each customer/client/ policy owner the transactions/policy payment from 2018 DEC to 2020 JUL were given in the dataset. So, **we divided the dataset into 3 segments which consist data of a customer for a period of 6 months. (Jan 2019- June 2019, July 2019- Dec 2019, Jan 2020- June 2020).**

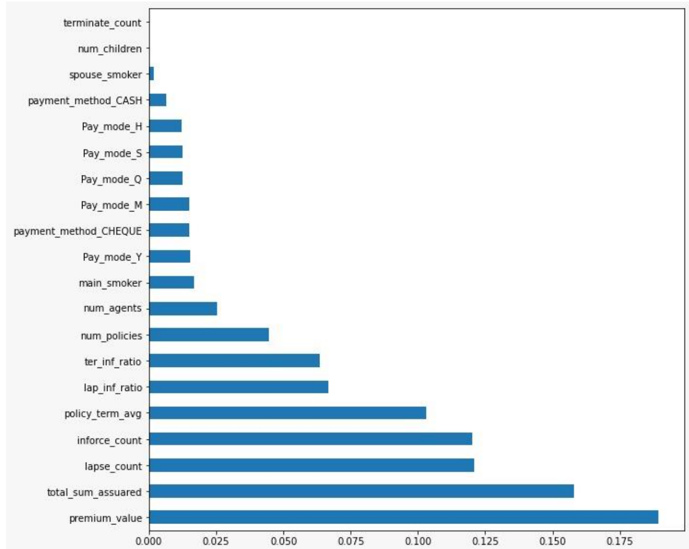


FIGURE 1: FEATURE IMPORTANCE FOR BEST PREDICTION MODEL FOR CROSS SELL USING RANDOM TREE CLASSIFIER

The strategy we followed was to train the model using the features extracted in previous 6 months and predict whether the customer will cross sell or not in the following 6 months. We extracted target variables (Cross-sell and Recommendation classes) for each segment using the data given for next 6 months.

Target variables:

Cross-sell: Whether the customer will cross sell in the next 6 months. This variable was extracted using **commencement_dt** and **policy_status** features in the dataset. For an example: target variable for Jan 2019 - June 2019 period data was extracted using the data in the period of July 2019- dec 2019.

Recommended Classes: This target variable was extracted by identifying new policies bought by the when he was cross selling in next 6 months. This variable was extracted using **commencement_dt, policy_status** and **product_name** features in the dataset.

According to the dataset we obtained, the lead indicators of cross selling are **payment mode (Annual, monthly, single payment..) , main smoker flag, spouse smoker flag, total sum assured, payment method (cash, cheque, both)** which are the given features and **policy term avg, avg premium value, lapse_count, terminate_count, lapse_inforce_ratio, terminate_inforce_ratio, number_of_children, number_of_policies, number_of _agents, total_sum_assured** which are the features synthesized using the given features.

Policy term avg: Average policy term in considered 6 months period

Avg_premium_value: Average premium value in considered 6 months period

Lapse_count: Number of times the customer lapse to pay the premium payment in considered 6 months period.

lapse_inforce_ratio: Ratio between lapsed count and in forced count within considered 6 months.

terminate_inforce_ratio: Ratio between lapsed count and terminated count within considered 6 months.

number_of_children: Number of children

number_of_policies: Number of policies maintained during the considered period.

number_of _agents: Number of agents assigned to a customer during considered period

total_sum_assured: Average total assured amount during considered period.

Below given image gives you the most important features selected. Those are: Premium value, total_sum_assured, lapse_count, inforce_count, policy_term_avg and so on.

Analytics Solution

Model methodology

Cross-selling prediction model

- Extracted features after each experiment was used to train different type of classifiers to identify best suitable model for the business problem. The classifiers considered during model building are as follows:
    - Logistic Regression
    - Decision Tree Classifier
    - XGBoost Classifier
    - Support Vector Classifier
    - Random Forest Classifier
    - Multi-Level Perceptron Classifier
    - Neural Networks
    - Extra Tree Classifier (ensemble approach)
- Hyper-tuning the parameters of each classifier was conducted during each experiment and best models were selected based on following performance metrics:
    - Macro F1 score
    - Validation accuracy
    - Precision score
    - Recall score.
    - Normalized confusion matrix.
    - AUC score
- Random Forest Classifier was selected as best models suitable for the problem.

- Performance metrics of our submissions are summarized in the following table:

Evaluation

Classifier	Training accuracy	Validation Accuracy	Precision Score	Recall Score	F1 Score (Validation)	AUC score
Random Forest classifier	0.9529	0.9368	0.5619	0.5994	0.5749	0.5995

** The best model was chosen using both the F-score and validation accuracy => Random Forest classifier

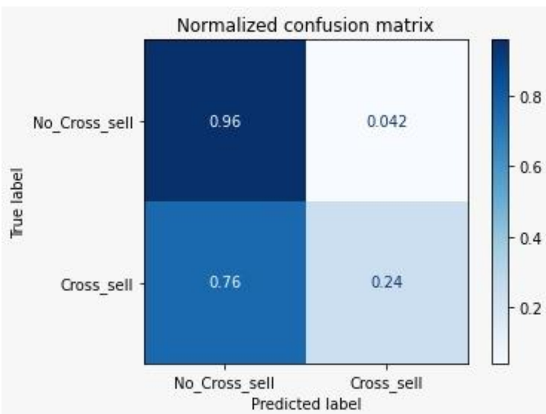


Figure 2 : Confusion Matrix for Best Prediction Model for Cross sell using Random Tree Classifier

```
Train accuracy : 0.9529468764249887
Validation accuracy : 0.936874385234045
Precision : 0.5619137345728114
Recall : 0.5994828464811957
F1-Score : 0.5749703725268034
Classification Report
              precision    recall  f1-score   support

           0       0.98      0.96      0.97     16777
           1       0.15      0.24      0.18       506

    accuracy                           0.94     17283
   macro avg       0.56      0.60      0.57     17283
weighted avg       0.95      0.94      0.94     17283

Confusion Matrix
[[16070   707]
 [  384   122]]
```

Figure 3 : Classification Report for Best Prediction Model for Cross sell using Random Tree Classifier

Recommendation model

Our idea to recommend suitable policy option is given below in the flow chart.

Both the features and recommendations were extracted using the approach which was explained previously. The cross sell (whether a customer will do cross sell or not?) was predicted with the aid of extracted features and Random Forest Classifier. Recommendation classes was extracted using the training data and was used along extracted features to predict suitable policy recommendation for each customer. The recommendation model was built using Random Forest Classifier and was tuned to improve its performance.

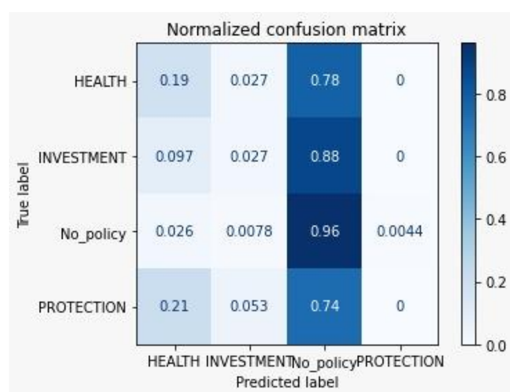


Figure 4: Confusion Matrix for Recommendation Model using Random Tree Classifier

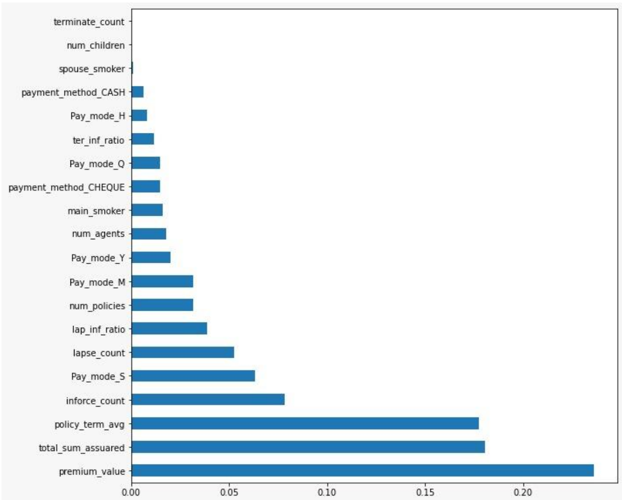
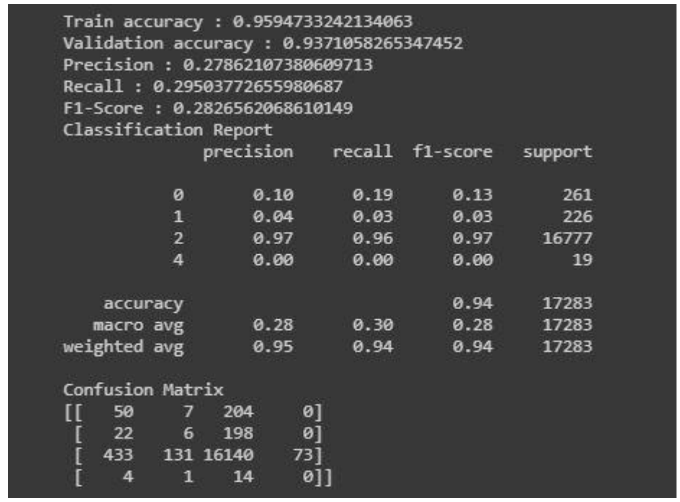


Figure 5: Feature Importance for Recommendation Model using Random Tree Classifier



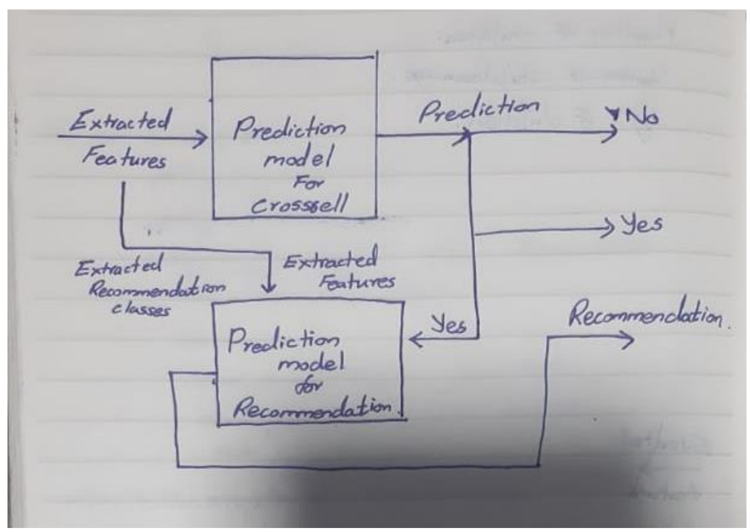
```
Train accuracy : 0.9594733242134063
Validation accuracy : 0.9371058265347452
Precision : 0.27862107380609713
Recall : 0.29503772655980687
F1-Score : 0.2826562068610149
Classification Report
              precision    recall  f1-score   support

           0       0.10      0.19      0.13       261
           1       0.04      0.03      0.03       226
           2       0.97      0.96      0.97     16777
           4       0.00      0.00      0.00        19

    accuracy                           0.94     17283
   macro avg       0.28      0.30      0.28     17283
weighted avg       0.95      0.94      0.94     17283

Confusion Matrix
[[   50     7   204     0]
 [   22     6   198     0]
 [  433   131 16140    73]
 [    4     1    14     0]]
```

Figure 6: Classification Report for Recommendation Model using Random Tree Classifier

Due to time constrain we were unable to hyper tune the recommendation model, where hyper tuning would have increased the accuracy of recommendation model.

Discussion

Key insights



The Insurance company's focus is to increase their revenue, which will get improved by both upsells and cross sells. Thus, the model they should employ should focus on accurately detecting the cross-sell probability to enhance their business. Accurate prediction can help agent to identify the potential cross selling customer and advertise/ promote him to go for a more expensive or advanced product. Also, accurate prediction will help agents to make targeted proactive interventions as there is a vast potential to increase cross sell rate.

The below features also should be extracted for efficient and accurate cross- selling prediction, but due to time constrain we were unable to extract the followings:

Age of main policy holder: Age is a good indicator to identify the type of recommendation that can be given. For example, if a policy owner's current age is above 55, then there is an opportunity for retirement policy plan, similarly for health policy plan. If the policy owner's age is around 18 then there is a higher chance for education policy plan.

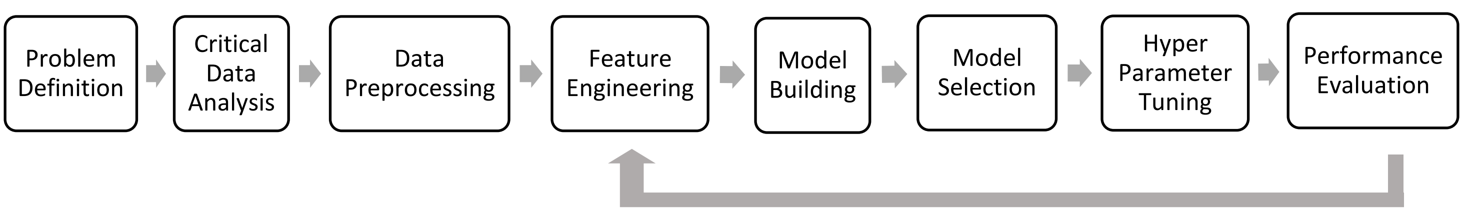
No of months as customer: Cross selling to an existing customer is lot easier and less expensive than gaining new customers only because those customers already know and trust the insurance. The number of months as customer feature is an indicator for the trust of policy owner on the insurance company.

Age of Children: Again, age of children can be key indicator to determine the type of recommendations. For example, if a child's age is less than or equal to 18, there is an opportunity for the policy owner to cross sell for an education policy on behalf of their children.

Interventions

- Identify the best target product for each customer
- Offer additional valuable services
- Performance review of agents
- Annually obtain customer feedbacks to maintain relationship
- Advertise/ promote – use social medias, phone calls
- Educate the customers – show them the benefits, understand their needs

Approaches



Tools Utilized

We used Google Colaboratory was utilized to build our experiments and it was selected because easy to share and it provides GPU and TPU features. The following libraries were utilized in our approach:

- Pandas
- NumPy
- Matplotlib
- Seaborn
- Imblearn
- Sklearn
- TensorFlow
- Keras

APPENDICES

```
**************************** feature extraction**********************************

# -*- coding: utf-8 -*-
"""DataStorm_Finals_Feature_Engineering.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1OTQm3_in1_wiDTiwhYI6t5E74vUf9jsu

##Import Libraries

###Neptune Ai
"""

! pip install neptune-client==0.4.132

pip install  neptune-contrib neptune-client

import neptune
from neptunecontrib.monitoring.keras import NeptuneMonitor
neptune.init(project_qualified_name='jathurshan0330/DataStorm2-finals', # change this to your
`workspace_name/project_name`

api_token='eyJhcGlfYWRkcmVzcyI6Imh0dHBzOi8vdWkubmVwdHVuZS5haSIsImFwaV91cmwiOiJodHRwc
zovL3VpLm5lcHR1bmUuYWkiLCJhcGlfa2V5IjoiZmJkZjYxNGYtMTA0ZC00ZTc1LWJiMTYtNzczNjgwZWQ3OT
UzIn0=', # change this to your api token
          )

"""### Other Libraries"""

import os
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.io
import seaborn as sns
from sklearn.model_selection import train_test_split
import tensorflow as tf
from scipy import stats
from tensorflow.keras.models import Sequential,Model
from tensorflow.keras.layers import Dense,
Input,LSTM,Reshape,Conv2D,Flatten,Dropout,BatchNormalization, LeakyReLU, concatenate, GRU,
GlobalMaxPooling1D, GlobalMaxPooling2D, Bidirectional

!pip install scikit-plot
```

```
"""##Read Data

###Mount Drive
"""

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

cd '/content/drive/My Drive/Datastorm2.0_Final'

!ls '/content/drive/My Drive/Datastorm2.0_Final'

"""###Data"""

agent_data = pd.read_csv('datastorm_agent_data.csv')
print(agent_data.head())
print(agent_data.shape)
print(agent_data.columns)

policy_data = pd.read_csv('datastorm_policy_data.csv')
print(policy_data.head())
print(policy_data.shape)
print(policy_data.columns)


test_data = pd.read_csv('testset.csv')
print(test_data.head())
print(test_data.shape)
print(test_data.columns)

print("Agent Data")
print(agent_data.isna().sum())
print('\n')
print("Policy Data")
print(policy_data.isna().sum())
print('\n')
print("Test Data")
print(test_data.isna().sum())

"""##Data Preprocessing

###One hot Encoding
"""

print(policy_data.shape)
print(policy_data.columns)
```

```
def getting_dummies(main_data,column_name, pre, drop = True):
  dummies = pd.get_dummies(main_data[column_name],drop_first=drop, prefix= pre)
  data = main_data.pop(column_name)
  main_data=pd.concat([main_data, dummies],axis=1)
  return main_data

policy_data = getting_dummies(policy_data, 'main_holder_gender', 'Gender',drop = True)

policy_data = getting_dummies(policy_data, 'policy_payment_mode', 'Pay_mode',drop = False)

policy_data = getting_dummies(policy_data, 'policy_status', 'status',drop = False)

policy_data = getting_dummies(policy_data, 'termination_reason', 'ter_reason',drop = False)

policy_data = getting_dummies(policy_data, 'main_holder_smoker_flag', 'main_smoker',drop = True)

policy_data = getting_dummies(policy_data, 'spouse_gender', 'Spouse_Gender',drop = False)

policy_data = getting_dummies(policy_data, 'spouse_smoker_flag', 'Spouse_smoker',drop = False)

policy_data = getting_dummies(policy_data, 'child1_gender', 'child1_gender',drop = False)

policy_data = getting_dummies(policy_data, 'child2_gender', 'child2_gender',drop = False)

policy_data = getting_dummies(policy_data, 'child3_gender', 'child3_gender',drop = False)

policy_data = getting_dummies(policy_data, 'child4_gender', 'child4_gender',drop = False)

policy_data = getting_dummies(policy_data, 'child5_gender', 'child5_gender',drop = False)

policy_data = getting_dummies(policy_data, 'payment_method', 'payment_method',drop = False)


print(policy_data.shape)
print(policy_data.columns)

"""###Data Analysis"""

client = policy_data["client_code"]
client = client.tolist()
client = set(client)
#client = client.tolist()
client_2 = []
for i in client:
  client_2.append(i)
print(len(client))
print(len(client_2))
```

```
from datetime import datetime
def days(start_date, end_date):
  start_date = datetime.strptime(start_date, "%Y/%m/%d")
  end_date = datetime.strptime(end_date, "%Y/%m/%d")
  #print((end_date - start_date).days)
  return (end_date - start_date).days

from collections import Counter
client_policies = [[] for i in range (len(client))]
client_policy_num = [[] for i in range (len(client))]
for i in range(len(policy_data)):
  a = policy_data['product_name'][i]
  b = policy_data['policy_code'][i]
  c = policy_data['client_code'][i]
  #print(policy_data['commencement_dt'][i])
  #day_ = days('2018/12/31', policy_data['commencement_dt'][i])
  ind = client_2.index(c)
  #print(c)
  #print(ind)
  #print(day_)
  #if len(client_policy_num[ind]) == 0:
   # if b not in client_policy_num[ind]:
    #  client_policies[ind].append(a)
     # client_policy_num[ind].append(b)

  #elif day_>=0:
  if b not in client_policy_num[ind]:
    client_policies[ind].append(a)
    client_policy_num[ind].append(b)
      #break

print(len(client_policy_num))
print(len(client_policies))
count = []
for i in client_policy_num:
  count.append(len(i))
print(Counter(count))

count = []
for i in client_policies:
  i = set(i)
  count.append(len(i))
print(Counter(count))

print(client_policy_num[14890])
print(client_policies[14890])
```

```
"""## Feature Engineering

###Breaking data into three time period
"""

# Sort Dataframe
policy_data_sorted = policy_data.sort_values(by=['client_code','policy_snapshot_as_on'], ignore_index
= True)

print(policy_data_sorted.shape)
print(policy_data_sorted['client_code'].head())
print(policy_data_sorted['policy_snapshot_as_on'].head())
policy_data_sorted.head()

policy_jan_jun_19 = []
policy_jul_dec_19 = []
policy_jan_jun_20 = []
column_names = policy_data_sorted.columns
first_6 = ['01','02','03','04','05','06']

for i in range(len(policy_data_sorted)):
  time = str(policy_data_sorted["policy_snapshot_as_on"][i])
  year = time[:4]
  month = time[4:6]
  date = time[6:]
  #print(time)
  #print(date)
  #print(month)
  #print(year)
  if year == '2019':
    if month in first_6:
      policy_jan_jun_19.append(policy_data_sorted.iloc[i])
    elif month == '07' and date == '01':
      policy_jan_jun_19.append(policy_data_sorted.iloc[i])
    else:
      policy_jul_dec_19.append(policy_data_sorted.iloc[i])
  elif year == '2020':
    if month =='01' and date == '01':
      policy_jul_dec_19.append(policy_data_sorted.iloc[i])
    else:
      policy_jan_jun_20.append(policy_data_sorted.iloc[i])
 # break


policy_jan_jun_19 = pd.DataFrame(policy_jan_jun_19)
policy_jan_jun_19 = policy_jan_jun_19.sort_values(by=['client_code','policy_snapshot_as_on'],
ignore_index = True)
```

```
policy_jul_dec_19 = pd.DataFrame(policy_jul_dec_19)
policy_jul_dec_19 = policy_jul_dec_19.sort_values(by=['client_code','policy_snapshot_as_on'],
ignore_index = True)
policy_jan_jun_20 = pd.DataFrame(policy_jan_jun_20)
policy_jan_jun_20 = policy_jan_jun_20.sort_values(by=['client_code','policy_snapshot_as_on'],
ignore_index = True)

print(policy_jan_jun_19.shape)
print(policy_jul_dec_19.shape)
print(policy_jan_jun_20.shape)

policy_jan_jun_19.head()

policy_jul_dec_19.head()

policy_jan_jun_20.head()

#Save to Drive
policy_jan_jun_19.to_csv('policy_jan_jun_19.csv',index=False)
policy_jul_dec_19.to_csv('policy_jul_dec_19.csv',index=False)
policy_jan_jun_20.to_csv('policy_jan_jun_20.csv',index=False)

a=[]
a.append(policy_data_sorted.iloc[0])
a.append(policy_data_sorted.iloc[1])
a=pd.DataFrame(a)
a.head()

"""###Feature Extraction

"""

# Read previously saved data
policy_jan_jun_19 = pd.read_csv('policy_jan_jun_19.csv')
print(policy_jan_jun_19.head())
print(policy_jan_jun_19.shape)

policy_jul_dec_19= pd.read_csv('policy_jul_dec_19.csv')
print(policy_jul_dec_19.head())
print(policy_jul_dec_19.shape)

policy_jan_jun_20 = pd.read_csv('policy_jan_jun_20.csv')
print(policy_jan_jun_20.head())
print(policy_jan_jun_20.shape)

data_jan_jun_19 = []
for i in range(len(policy_jan_jun_19)):
  x = policy_jan_jun_19['client_code'][i]
```

```
  if x not in data_jan_jun_19:
    data_jan_jun_19.append(x)
data_jan_jun_19 = pd.DataFrame(data_jan_jun_19, columns = ['client_code'])

print(len(data_jan_jun_19))
print(data_jan_jun_19.head())

data_jul_dec_19 = []
for i in range(len(policy_jul_dec_19)):
  x = policy_jul_dec_19['client_code'][i]
  if x not in data_jul_dec_19:
    data_jul_dec_19.append(x)
data_jul_dec_19 = pd.DataFrame(data_jul_dec_19, columns = ['client_code'])

print(len(data_jul_dec_19))
print(data_jul_dec_19.head())

data_jan_jun_20 = []
for i in range(len(policy_jan_jun_20)):
  x = policy_jan_jun_20['client_code'][i]
  if x not in data_jan_jun_20:
    data_jan_jun_20.append(x)
data_jan_jun_20 = pd.DataFrame(data_jan_jun_20, columns = ['client_code'])

print(len(data_jan_jun_20))
print(data_jan_jun_20.head())

from datetime import datetime
def days(start_date, end_date):
  start_date = datetime.strptime(start_date, "%Y/%m/%d")
  end_date = datetime.strptime(end_date, "%Y/%m/%d")
  #print((end_date - start_date).days)
  return (end_date - start_date).days

"""####Extracting Labels"""

#Extracting Labels for jan to june 2019 using jul to dec data
labels_jan_jun_19 = []
recom_jan_jun_19 = []
for i in range (len(data_jan_jun_19)):
  temp = policy_jul_dec_19[policy_jul_dec_19['client_code'] == data_jan_jun_19['client_code'][i]]
  temp = temp.sort_values(by=['client_code','policy_snapshot_as_on'], ignore_index = True)
  #print(temp)
  inforce_count = 0
  new_policy_count = 0
  new_policy = 'No_policy'
  for j in range (len(temp)):
    #print(temp['commencement_dt'][j])
```

```
    day_ = days('2019/06/30', temp['commencement_dt'][j])
    day_2 = days('2019/12/31', temp['commencement_dt'][j])
    #print(day_)
    if (temp['status_INFORCE'][j] == 1 or temp['status_LAPSED'][j] == 1)  and day_ <= 0:
      inforce_count+=1
    if (temp['status_INFORCE'][j] == 1 or temp['status_LAPSED'][j] == 1)  and day_ > 0 and day_2 <= 0:
      new_policy_count+=1
      new_policy = temp['product_name'][j]
    if inforce_count >=1 and new_policy_count >=1 :
      break

  if inforce_count > 0 and new_policy_count > 0:
    labels_jan_jun_19.append(1)
    recom_jan_jun_19.append(new_policy)
  else:
    labels_jan_jun_19.append(0)
    recom_jan_jun_19.append('No_policy')

labels_jan_jun_19 = pd.DataFrame(labels_jan_jun_19, columns = ['cross_sell'])
labels_jan_jun_19 = pd.concat([data_jan_jun_19,labels_jan_jun_19],axis=1)

recom_jan_jun_19 = pd.DataFrame(recom_jan_jun_19, columns = ['recommentation'])
labels_jan_jun_19 = pd.concat([labels_jan_jun_19, recom_jan_jun_19],axis=1)

print(labels_jan_jun_19.shape)
labels_jan_jun_19.head()

print(Counter(labels_jan_jun_19['cross_sell']))
print(Counter(labels_jan_jun_19['recommentation']))

labels_jan_jun_19.to_csv('labels_jan_jun_19.csv',index=False)

#Extracting Labels for july to dec 2019 using jan to june 2020 data
labels_jul_dec_19 = []
recom_jul_dec_19 = []
for i in range (len(data_jul_dec_19)):
  temp = policy_jan_jun_20[policy_jan_jun_20['client_code'] == data_jul_dec_19['client_code'][i]]
  temp = temp.sort_values(by=['client_code','policy_snapshot_as_on'], ignore_index = True)
  #print(temp)
  inforce_count = 0
  new_policy_count = 0
  new_policy = 'No_policy'
  for j in range (len(temp)):
    #print(temp['commencement_dt'][j])
    day_ = days('2019/12/31', temp['commencement_dt'][j])
    day_2 = days('2020/6/30', temp['commencement_dt'][j])
    #print(day_)
    if (temp['status_INFORCE'][j] == 1 or temp['status_LAPSED'][j] == 1)  and day_ <= 0:
```

```
      inforce_count+=1
    if (temp['status_INFORCE'][j] == 1 or temp['status_LAPSED'][j] == 1)  and day_ > 0 and day_2 <= 0:
      new_policy_count+=1
      new_policy = temp['product_name'][j]
    if inforce_count >=1 and new_policy_count >=1 :
      break

  if inforce_count > 0 and new_policy_count > 0:
    labels_jul_dec_19.append(1)
    recom_jul_dec_19.append(new_policy)
  else:
    labels_jul_dec_19.append(0)
    recom_jul_dec_19.append('No_policy')

labels_jul_dec_19 = pd.DataFrame(labels_jul_dec_19, columns = ['cross_sell'])
labels_jul_dec_19 = pd.concat([data_jul_dec_19,labels_jul_dec_19],axis=1)

recom_jul_dec_19 = pd.DataFrame(recom_jul_dec_19, columns = ['recommentation'])
labels_jul_dec_19 = pd.concat([labels_jul_dec_19, recom_jul_dec_19],axis=1)

print(labels_jul_dec_19.shape)
labels_jul_dec_19.head()

print(Counter(labels_jul_dec_19['cross_sell']))
print(Counter(labels_jul_dec_19['recommentation']))

labels_jul_dec_19.to_csv('labels_jul_dec_19.csv',index=False)

"""####Features"""

from datetime import datetime
def age(date,date2):
  year = datetime.strptime(date, "%Y/%m/%d").year
  new_year = datetime.strptime(date2, '%Y%m%d').year
  return new_year-year

# extracting features for jan to june
# policy_term_avg, Pay_mode_H,           Pay_mode_M,   Pay_mode_Q,    Pay_mode_S,     Pay_mode_Y,
main_smoker, spouse_smoker, inforce_count, lapse_count, terminate_count, lap_inf_ratio,
ter_inf_ratio, num_children, num_policies, num_agents, total_sum_assuared, premium_value,
payment_method_CASH, payment_method_CHEQUE
import datetime
features_jan_jun_2019 = []

for i in range (len(data_jan_jun_19)):

  temp = policy_jan_jun_19[policy_jan_jun_19['client_code'] == data_jan_jun_19['client_code'][i]]
  temp = temp.sort_values(by=['client_code','policy_snapshot_as_on'], ignore_index = True)
```

```
# policy_term_avg
policy_term_avg = temp['policy_term']
policy_term_avg = np.mean(np.array(policy_term_avg))
#print(policy_term_avg)
h = 0
m = 0
q = 0
s = 0
y = 0
smoke = 0
spouse_smoke = 0
cash = 0
cheque = 0

inforce_count = temp['status_INFORCE']
inforce_count = np.sum(np.array(inforce_count))

lapse_count = temp['status_LAPSED']
lapse_count = np.sum(np.array(lapse_count))


terminate_count = 0

for j in range(len(temp)):
  if temp['Pay_mode_H'][j] == 1:
    h = 1
  if temp['Pay_mode_M'][j] == 1:
    m = 1
  if temp['Pay_mode_Q'][j] == 1:
    q = 1
  if temp['Pay_mode_S'][j] == 1:
    s = 1
  if temp['Pay_mode_Y'][j] == 1:
    y = 1
  if temp['main_smoker_Y'][j] == 1:
    smoke = 1
  if temp['Spouse_smoker_Y'][j] == 1:
    spouse_smoke = 1
  if temp['payment_method_CASH'][j] == 1:
    cash = 1
  if temp['payment_method_CHEQUE'][j] == 1:
    cheque = 1

  if isinstance(temp['termination_dt'][j], datetime.datetime):
    day_ter =  days('2018/12/31', temp['termination_dt'][j])
    day_ter_2 =  days('2019/6/30', temp['termination_dt'][j])
    if day_ter > 0 and day_ter_2 <= 0:
```

```
      terminate_count+=1

  if inforce_count == 0:
    lap_inf_ratio = 1
  else:
    lap_inf_ratio = lapse_count/inforce_count

  if inforce_count == 0:
    ter_inf_ratio = 1
  else:
    ter_inf_ratio = terminate_count/inforce_count

  num_children = 0

  if isinstance(temp['child1_dob'][len(temp)-1], datetime.datetime) == False:
    num_children +=1
  if isinstance(temp['child2_dob'][len(temp)-1], datetime.datetime) == False:
    num_children +=1
  if isinstance(temp['child3_dob'][len(temp)-1], datetime.datetime) == False:
    num_children +=1
  if isinstance(temp['child4_dob'][len(temp)-1], datetime.datetime) == False:
    num_children +=1
  if isinstance(temp['child5_dob'][len(temp)-1], datetime.datetime) == False:
    num_children +=1

  num_policies = temp['policy_code']
  num_policies = num_policies.tolist()
  num_policies = set(num_policies)
  num_policies = len(num_policies)

  num_agents = temp['agent_code']
  num_agents = num_agents.tolist()
  num_agents = set(num_agents)
  num_agents = len(num_agents)

  total_sum_assuared_avg = temp['total_sum_assuared']
  total_sum_assuared_avg = np.mean(np.array(total_sum_assuared_avg))
  #print(total_sum_assuared_avg)

  premium_value_avg = temp['premium_value']
  premium_value_avg = np.mean(np.array(premium_value_avg))
  #print(premium_value_avg)

  feat = [policy_term_avg, h, m, q, s, y, smoke, spouse_smoke, inforce_count, lapse_count,
terminate_count, lap_inf_ratio, ter_inf_ratio, num_children, num_policies, num_agents,
total_sum_assuared_avg, premium_value_avg, cash, cheque]
 # policy_term_avg, Pay_mode_H,          Pay_mode_M,   Pay_mode_Q,    Pay_mode_S,     Pay_mode_Y,
main_smoker, spouse_smoker, inforce_count, lapse_count, terminate_count, lap_inf_ratio,
```

```
ter_inf_ratio, num_children, num_policies, num_agents, total_sum_assuared, premium_value,
payment_method_CASH, payment_method_CHEQUE

  #print(feat)
  features_jan_jun_2019.append(feat)
  #break

column_names = ['policy_term_avg', 'Pay_mode_H',       'Pay_mode_M', 'Pay_mode_Q',  'Pay_mode_S',
           'Pay_mode_Y', 'main_smoker', 'spouse_smoker', 'inforce_count', 'lapse_count',
'terminate_count', 'lap_inf_ratio', 'ter_inf_ratio', 'num_children', 'num_policies', 'num_agents',
'total_sum_assuared', 'premium_value', 'payment_method_CASH', 'payment_method_CHEQUE']

features_jan_jun_2019 = pd.DataFrame(features_jan_jun_2019,columns=column_names)




print(features_jan_jun_2019.shape)
features_jan_jun_2019.head()

train_data_jan_jun_19 = pd.concat([data_jan_jun_19,features_jan_jun_2019],axis=1)
train_data_jan_jun_19.head()

train_data_jan_jun_19.to_csv('train_data_jan_jun_19.csv',index=False)



# extracting features for july to dec 2019
# policy_term_avg, Pay_mode_H,          Pay_mode_M,   Pay_mode_Q,    Pay_mode_S,     Pay_mode_Y,
main_smoker, spouse_smoker, inforce_count, lapse_count, terminate_count, lap_inf_ratio,
ter_inf_ratio, num_children, num_policies, num_agents, total_sum_assuared, premium_value,
payment_method_CASH, payment_method_CHEQUE
import datetime
features_jul_dec_2019 = []

for i in range (len(data_jul_dec_19)):

  temp = policy_jul_dec_19[policy_jul_dec_19['client_code'] == data_jul_dec_19['client_code'][i]]
  temp = temp.sort_values(by=['client_code','policy_snapshot_as_on'], ignore_index = True)

  # policy_term_avg
  policy_term_avg = temp['policy_term']
  policy_term_avg = np.mean(np.array(policy_term_avg))
  #print(policy_term_avg)
  h = 0
  m = 0
  q = 0
  s = 0
```

```
y = 0
smoke = 0
spouse_smoke = 0
cash = 0
cheque = 0

inforce_count = temp['status_INFORCE']
inforce_count = np.sum(np.array(inforce_count))

lapse_count = temp['status_LAPSED']
lapse_count = np.sum(np.array(lapse_count))


terminate_count = 0

for j in range(len(temp)):
  if temp['Pay_mode_H'][j] == 1:
    h = 1
  if temp['Pay_mode_M'][j] == 1:
    m = 1
  if temp['Pay_mode_Q'][j] == 1:
    q = 1
  if temp['Pay_mode_S'][j] == 1:
    s = 1
  if temp['Pay_mode_Y'][j] == 1:
    y = 1
  if temp['main_smoker_Y'][j] == 1:
    smoke = 1
  if temp['Spouse_smoker_Y'][j] == 1:
    spouse_smoke = 1
  if temp['payment_method_CASH'][j] == 1:
    cash = 1
  if temp['payment_method_CHEQUE'][j] == 1:
    cheque = 1

  if isinstance(temp['termination_dt'][j], datetime.datetime):
    day_ter =  days('2019/06/30', temp['termination_dt'][j])
    day_ter_2 =  days('2019/12/31', temp['termination_dt'][j])
    if day_ter > 0 and day_ter_2 <= 0:
      terminate_count+=1

if inforce_count == 0:
  lap_inf_ratio = 1
else:
  lap_inf_ratio = lapse_count/inforce_count

if inforce_count == 0:
  ter_inf_ratio = 1
```

```
  else:
    ter_inf_ratio = terminate_count/inforce_count

  num_children = 0

  if isinstance(temp['child1_dob'][len(temp)-1], datetime.datetime) == False:
    num_children +=1
  if isinstance(temp['child2_dob'][len(temp)-1], datetime.datetime) == False:
    num_children +=1
  if isinstance(temp['child3_dob'][len(temp)-1], datetime.datetime) == False:
    num_children +=1
  if isinstance(temp['child4_dob'][len(temp)-1], datetime.datetime) == False:
    num_children +=1
  if isinstance(temp['child5_dob'][len(temp)-1], datetime.datetime) == False:
    num_children +=1

  num_policies = temp['policy_code']
  num_policies = num_policies.tolist()
  num_policies = set(num_policies)
  num_policies = len(num_policies)

  num_agents = temp['agent_code']
  num_agents = num_agents.tolist()
  num_agents = set(num_agents)
  num_agents = len(num_agents)

  total_sum_assuared_avg = temp['total_sum_assuared']
  total_sum_assuared_avg = np.mean(np.array(total_sum_assuared_avg))
  #print(total_sum_assuared_avg)

  premium_value_avg = temp['premium_value']
  premium_value_avg = np.mean(np.array(premium_value_avg))
  #print(premium_value_avg)

  feat = [policy_term_avg, h, m, q, s, y, smoke, spouse_smoke, inforce_count, lapse_count,
terminate_count, lap_inf_ratio, ter_inf_ratio, num_children, num_policies, num_agents,
total_sum_assuared_avg, premium_value_avg, cash, cheque]
 # policy_term_avg, Pay_mode_H,          Pay_mode_M,   Pay_mode_Q,    Pay_mode_S,     Pay_mode_Y,
main_smoker, spouse_smoker, inforce_count, lapse_count, terminate_count, lap_inf_ratio,
ter_inf_ratio, num_children, num_policies, num_agents, total_sum_assuared, premium_value,
payment_method_CASH, payment_method_CHEQUE

  #print(feat)
  features_jul_dec_2019.append(feat)
  #break

column_names = ['policy_term_avg', 'Pay_mode_H',       'Pay_mode_M', 'Pay_mode_Q',  'Pay_mode_S',
          'Pay_mode_Y', 'main_smoker', 'spouse_smoker', 'inforce_count', 'lapse_count',
```

```
'terminate_count', 'lap_inf_ratio', 'ter_inf_ratio', 'num_children', 'num_policies', 'num_agents',
'total_sum_assuared', 'premium_value', 'payment_method_CASH', 'payment_method_CHEQUE']

features_jul_dec_2019 = pd.DataFrame(features_jul_dec_2019,columns=column_names)




print(features_jul_dec_2019.shape)
features_jul_dec_2019.head()

train_data_jul_dec_19 = pd.concat([data_jul_dec_19,features_jul_dec_2019],axis=1)
train_data_jul_dec_19.head()

train_data_jul_dec_19.to_csv('train_data_jul_dec_19.csv',index=False)



# extracting features for jan to jun 2020
# policy_term_avg, Pay_mode_H,          Pay_mode_M,   Pay_mode_Q,   Pay_mode_S,    Pay_mode_Y,
main_smoker, spouse_smoker, inforce_count, lapse_count, terminate_count, lap_inf_ratio,
ter_inf_ratio, num_children, num_policies, num_agents, total_sum_assuared, premium_value,
payment_method_CASH, payment_method_CHEQUE
import datetime
features_jan_jun_2020 = []

for i in range (len(data_jan_jun_20)):

  temp = policy_jan_jun_20[policy_jan_jun_20['client_code'] == data_jan_jun_20['client_code'][i]]
  temp = temp.sort_values(by=['client_code','policy_snapshot_as_on'], ignore_index = True)

  # policy_term_avg
  policy_term_avg = temp['policy_term']
  policy_term_avg = np.mean(np.array(policy_term_avg))
  #print(policy_term_avg)
  h = 0
  m = 0
  q = 0
  s = 0
  y = 0
  smoke = 0
  spouse_smoke = 0
  cash = 0
  cheque = 0

  inforce_count = temp['status_INFORCE']
  inforce_count = np.sum(np.array(inforce_count))
```

```
lapse_count = temp['status_LAPSED']
lapse_count = np.sum(np.array(lapse_count))


terminate_count = 0

for j in range(len(temp)):
  if temp['Pay_mode_H'][j] == 1:
    h = 1
  if temp['Pay_mode_M'][j] == 1:
    m = 1
  if temp['Pay_mode_Q'][j] == 1:
    q = 1
  if temp['Pay_mode_S'][j] == 1:
    s = 1
  if temp['Pay_mode_Y'][j] == 1:
    y = 1
  if temp['main_smoker_Y'][j] == 1:
    smoke = 1
  if temp['Spouse_smoker_Y'][j] == 1:
    spouse_smoke = 1
  if temp['payment_method_CASH'][j] == 1:
    cash = 1
  if temp['payment_method_CHEQUE'][j] == 1:
    cheque = 1

  if isinstance(temp['termination_dt'][j], datetime.datetime):
    day_ter =  days('2019/12/31', temp['termination_dt'][j])
    day_ter_2 =  days('2020/6/30', temp['termination_dt'][j])
    if day_ter > 0 and day_ter_2 <= 0:
      terminate_count+=1

if inforce_count == 0:
  lap_inf_ratio = 1
else:
  lap_inf_ratio = lapse_count/inforce_count

if inforce_count == 0:
  ter_inf_ratio = 1
else:
  ter_inf_ratio = terminate_count/inforce_count

num_children = 0

if isinstance(temp['child1_dob'][len(temp)-1], datetime.datetime) == False:
  num_children +=1
if isinstance(temp['child2_dob'][len(temp)-1], datetime.datetime) == False:
  num_children +=1
```

```
  if isinstance(temp['child3_dob'][len(temp)-1], datetime.datetime) == False:
    num_children +=1
  if isinstance(temp['child4_dob'][len(temp)-1], datetime.datetime) == False:
    num_children +=1
  if isinstance(temp['child5_dob'][len(temp)-1], datetime.datetime) == False:
    num_children +=1

  num_policies = temp['policy_code']
  num_policies = num_policies.tolist()
  num_policies = set(num_policies)
  num_policies = len(num_policies)

  num_agents = temp['agent_code']
  num_agents = num_agents.tolist()
  num_agents = set(num_agents)
  num_agents = len(num_agents)

  total_sum_assuared_avg = temp['total_sum_assuared']
  total_sum_assuared_avg = np.mean(np.array(total_sum_assuared_avg))
  #print(total_sum_assuared_avg)

  premium_value_avg = temp['premium_value']
  premium_value_avg = np.mean(np.array(premium_value_avg))
  #print(premium_value_avg)

  feat = [policy_term_avg, h, m, q, s, y, smoke, spouse_smoke, inforce_count, lapse_count,
terminate_count, lap_inf_ratio, ter_inf_ratio, num_children, num_policies, num_agents,
total_sum_assuared_avg, premium_value_avg, cash, cheque]
 # policy_term_avg, Pay_mode_H,          Pay_mode_M,   Pay_mode_Q,    Pay_mode_S,     Pay_mode_Y,
main_smoker, spouse_smoker, inforce_count, lapse_count, terminate_count, lap_inf_ratio,
ter_inf_ratio, num_children, num_policies, num_agents, total_sum_assuared, premium_value,
payment_method_CASH, payment_method_CHEQUE

  #print(feat)
  features_jan_jun_2020.append(feat)
  #break

column_names = ['policy_term_avg', 'Pay_mode_H',       'Pay_mode_M', 'Pay_mode_Q',  'Pay_mode_S',
            'Pay_mode_Y', 'main_smoker', 'spouse_smoker', 'inforce_count', 'lapse_count',
'terminate_count', 'lap_inf_ratio', 'ter_inf_ratio', 'num_children', 'num_policies', 'num_agents',
'total_sum_assuared', 'premium_value', 'payment_method_CASH', 'payment_method_CHEQUE']

features_jan_jun_2020 = pd.DataFrame(features_jan_jun_2020,columns=column_names)




print(features_jan_jun_2020.shape)
```

```
features_jan_jun_2020.head()

test_data_jan_jun_20 = pd.concat([data_jan_jun_20,features_jan_jun_2020],axis=1)
test_data_jan_jun_20.head()

test_data_jan_jun_20.head()

test_data_jan_jun_20.to_csv('test_data_jan_jun_20.csv',index=False)
```

**************** Model building**********************

```
# -*- coding: utf-8 -*-
"""DataStorm2.0_Finals_Model_Building.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1MQbDT480-ALY1AbdxiqVIWPBXKYDtaVM

#Import Libraries

##Neptune Ai
"""

! pip install neptune-client==0.4.132

pip install  neptune-contrib neptune-client

import neptune
from neptunecontrib.monitoring.keras import NeptuneMonitor
neptune.init(project_qualified_name='jathurshan0330/DataStorm2-round1', # change this to your
`workspace_name/project_name`

api_token='eyJhcGlfYWRkcmVzcyI6Imh0dHBzOi8vdWkubmVwdHVuZS5haSIsImFwaV91cmwiOiJodHRwc
zovL3VpLm5lcHR1bmUuYWkiLCJhcGlfa2V5IjoiZmJkZjYxNGYtMTA0ZC00ZTc1LWJiMTYtNzczNjgwZWQ3OT
UzIn0=', # change this to your api token
          )

"""##other necessary libraries"""

import os
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.io
import seaborn as sns
from sklearn.model_selection import train_test_split
import tensorflow as tf
from scipy import stats
```

```
from tensorflow.keras.models import Sequential,Model
from tensorflow.keras.layers import Dense,
Input,LSTM,Reshape,Conv2D,Flatten,Dropout,BatchNormalization, LeakyReLU, concatenate, GRU,
GlobalMaxPooling1D, GlobalMaxPooling2D, Bidirectional

!pip install scikit-plot

"""#Read Data

##Mount Drive
"""

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

cd '/content/drive/My Drive/Datastorm2.0_Final'

!ls '/content/drive/My Drive/Datastorm2.0_Final'

"""##Data"""

train_data_1 = pd.read_csv('train_data_jan_jun_19.csv')
print(train_data_1.head())
print(train_data_1.shape)
train_data_2 = pd.read_csv('train_data_jul_dec_19.csv')
print(train_data_2.head())
print(train_data_2.shape)

train_labels_1 = pd.read_csv('labels_jan_jun_19.csv')
train_labels_2 = pd.read_csv('labels_jul_dec_19.csv')

test_data = pd.read_csv('test_data_jan_jun_20.csv')
print(test_data.head())
print(test_data.shape)


test = pd.read_csv('testset.csv')
test_3 = test.tolist()

train_data = pd.concat([train_data_1, train_data_2],axis=0)
print(train_data.shape)
train_label = pd.concat([train_labels_1, train_labels_2],axis=0)
print(train_label.shape)

from sklearn.model_selection import train_test_split
train_data, val_data, train_label, val_label = train_test_split(train_data, train_label, test_size=0.33,
random_state=42)
```

```
"""#Model Building"""

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score, roc_auc_score,
classification_report, plot_confusion_matrix, precision_score, recall_score
from sklearn import tree, svm
from sklearn.ensemble import RandomForestClassifier
import xgboost
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import NearestNeighbors,KNeighborsClassifier

print(train_data.shape)
print(val_data.shape)
print(test_data.shape)

"""##logistic regression approach"""

model= LogisticRegression(multi_class='multinomial', solver='saga',max_iter=100)
model.fit(train_data,train_label)
y_predict=model.predict(val_data)
print("Train accuracy : "+str(model.score(train_data,train_label)))
print("Validation accuracy : "+str(model.score(val_data,val_label)))
print("Precision : "+str(precision_score(val_label,y_predict,average='macro', zero_division=0)))
print("Recall : "+str(recall_score(val_label,y_predict,average='macro', zero_division=0)))
print("F1-Score : "+str(f1_score(val_label,y_predict,average='macro', zero_division=0)))
print("Classification Report")
print(classification_report(val_label,y_predict,zero_division=0))
print("Confusion Matrix")
print(confusion_matrix(val_label,y_predict))
#fig, ax = plt.subplots()
#plot_confusion_matrix(val_label, y_predict, ax=ax)


#neptune.log_metric('Training Accuracy', model.score(train_data,train_label))
#neptune.log_metric('Validation Accuracy', model.score(val_data,val_label))
#neptune.log_metric('Precision',precision_score(val_label,y_predict,average='macro', zero_division=0))
#neptune.log_metric('Recall', recall_score(val_label,y_predict,average='macro', zero_division=0))
#neptune.log_metric('F1-Score',f1_score(val_label,y_predict,average='macro', zero_division=0))

"""##Decision Tree Classifier model approach"""

#model = DecisionTreeClassifier(max_depth=20, class_weight = 'balanced' ).fit(train_data,train_label)
#model = DecisionTreeClassifier( max_depth = 20, class_weight = 'balanced', max_features = 'log2',
random_state = 8  ).fit(train_data,train_label) #hypertuned for all features
#model = DecisionTreeClassifier( max_depth = 21, class_weight = 'balanced', max_features = 'log2',
random_state = 31  ).fit(train_data,train_label) #hypertuned for 28 features
```

```
model = DecisionTreeClassifier(max_depth = None, min_samples_split=17, class_weight = 'balanced',
max_features = 'log2', random_state = 29  ).fit(train_data,train_label) #hypertuned for 28 features
without max depth
#model = DecisionTreeClassifier( max_depth = 21, class_weight = 'balanced', max_features = 'log2',
random_state = 11  ).fit(train_data,train_label) #hypertuned for 10 features


y_predict=model.predict(val_data)
print("Train accuracy : "+str(model.score(train_data,train_label)))
print("Validation accuracy : "+str(model.score(val_data,val_label)))
print("Precision : "+str(precision_score(val_label,y_predict,average='macro', zero_division=0)))
print("Recall : "+str(recall_score(val_label,y_predict,average='macro', zero_division=0)))
print("F1-Score : "+str(f1_score(val_label,y_predict,average='macro', zero_division=0)))
print("Classification Report")
print(classification_report(val_label,y_predict,zero_division=0))
print("Confusion Matrix")
print(confusion_matrix(val_label,y_predict))
#fig, ax = plt.subplots()
#plot_confusion_matrix(val_label, y_predict, ax=ax)


#neptune.log_metric('Training Accuracy', model.score(train_data,train_label))
#neptune.log_metric('Validation Accuracy', model.score(val_data,val_label))
#neptune.log_metric('Precision',precision_score(val_label,y_predict,average='macro', zero_division=0))
#neptune.log_metric('Recall', recall_score(val_label,y_predict,average='macro', zero_division=0))
#neptune.log_metric('F1-Score',f1_score(val_label,y_predict,average='macro', zero_division=0))

# Plot non-normalized confusion matrix
titles_options = [("Confusion matrix, without normalization", None),
                  ("Normalized confusion matrix", 'true')]
class_names = ["Check-In", "Canceled", "No-Show"]
for title, normalize in titles_options:
    disp = plot_confusion_matrix(model, val_data, val_label,
                                 display_labels=class_names,
                                 cmap=plt.cm.Blues,
                                 normalize=normalize)
    disp.ax_.set_title(title)

    print(title)
    print(disp.confusion_matrix)

#Feature Importance in Decision Tree Classifier
print("Feature Importance")
print(model.feature_importances_) #use inbuilt class feature_importances of tree based classifiers
#plot graph of feature importances for better visualization
plt.figure(figsize=[10,10])
feat_importances = pd.Series(model.feature_importances_, index=train_data.columns)
feat_importances.nlargest(47).plot(kind='barh')
```

```
plt.show()
#print(feat_importances)
results=pd.DataFrame()
results['columns']=train_data.columns
results['importances'] = model.feature_importances_
results.sort_values(by='importances',ascending=False,inplace=True)

results[:44]
selected_features = results['columns'][:28].tolist()
print(selected_features)
print(len(selected_features))

selected_features = ['reserve_duration', 'tot_cost', 'Room_Rate', 'tot_cost_per_day', 'Age',
'Discount_Rate', 'Adults', 'Children', 'Meal_BB', 'stay_duration', 'Babies', 'Coun_South', 'Edu_College',
'In_below25K', 'Promo_Yes', 'Gen_M', 'week_end', 'In_25K_50K', 'Visit_Yes', 'Dep_Refundable',
'Book_Online', 'Coun_North', 'Eth_Latino', 'Dep_No Deposit', 'Edu_Grad', 'Book_Direct', 'Car_Yes',
'In_50K_100K']

"""##XGB Boost Approach"""

clf = DecisionTreeClassifier(max_depth=50, class_weight = 'balanced')
#clf = DecisionTreeClassifier( max_depth = 50, class_weight = 'balanced', max_features = 'log2',
random_state = 8  )
#model=xgboost.XGBClassifier(base_estimator=clf,max_depth=20,n_estimators=15,objective='multi:sof
tmax',gamma=4.63,learning_rate=0.2,reg_lambda=1).fit(train_data,train_label) # day 2 second
submission model
model=xgboost.XGBClassifier(base_estimator = clf, max_depth = 22, n_estimators = 15, objective =
'multi:softmax', gamma = 4.5, learning_rate = 0.05, reg_lambda = 3.4).fit(train_data,train_label)
#hypertuned model
#model=xgboost.XGBClassifier(base_estimator = clf, max_depth = 19, n_estimators = 15, objective =
'multi:softmax', gamma = 4.5, learning_rate = 0.24, reg_lambda = 3.4).fit(train_data,train_label)
#hypertuned model

y_predict=model.predict(val_data)
print("Train accuracy : "+str(model.score(train_data,train_label)))
print("Validation accuracy : "+str(model.score(val_data,val_label)))
print("Precision : "+str(precision_score(val_label,y_predict,average='macro', zero_division=0)))
print("Recall : "+str(recall_score(val_label,y_predict,average='macro', zero_division=0)))
print("F1-Score : "+str(f1_score(val_label,y_predict,average='macro', zero_division=0)))
print("Classification Report")
print(classification_report(val_label,y_predict,zero_division=0))
print("Confusion Matrix")
print(confusion_matrix(val_label,y_predict))
#fig, ax = plt.subplots()
#plot_confusion_matrix(val_label, y_predict, ax=ax)
```

```
#neptune.log_metric('Training Accuracy', model.score(train_data,train_label))
#neptune.log_metric('Validation Accuracy', model.score(val_data,val_label))
#neptune.log_metric('Precision',precision_score(val_label,y_predict,average='macro', zero_division=0))
#neptune.log_metric('Recall', recall_score(val_label,y_predict,average='macro', zero_division=0))
#neptune.log_metric('F1-Score',f1_score(val_label,y_predict,average='macro', zero_division=0))

# Plot non-normalized confusion matrix
titles_options = [("Confusion matrix, without normalization", None),
                  ("Normalized confusion matrix", 'true')]
class_names = ["Check-In", "Canceled", "No-Show"]
for title, normalize in titles_options:
    disp = plot_confusion_matrix(model, val_data, val_label,
                                 display_labels=class_names,
                                 cmap=plt.cm.Blues,
                                 normalize=normalize)
    disp.ax_.set_title(title)

    print(title)
    print(disp.confusion_matrix)

#Feature Importance in XGBoost
print("Feature Importance")
print(model.feature_importances_) #use inbuilt class feature_importances of tree based classifiers
#plot graph of feature importances for better visualization
plt.figure(figsize=[10,10])
feat_importances = pd.Series(model.feature_importances_, index=train_data.columns)
feat_importances.nlargest(40).plot(kind='barh')

plt.show()
#print(feat_importances)
results=pd.DataFrame()
results['columns']=train_data.columns
results['importances'] = model.feature_importances_
results.sort_values(by='importances',ascending=False,inplace=True)

results[2:]
selected_features = results['columns'][2:].tolist()
print(selected_features)

"""##Support Vector Machine Approach"""

model = svm.SVC(degree=5,decision_function_shape='ovo', class_weight = 'balanced')
model.fit(train_data,train_label)
y_predict=model.predict(val_data)
print("Train accuracy : "+str(model.score(train_data,train_label)))
print("Validation accuracy : "+str(model.score(val_data,val_label)))
print("Precision : "+str(precision_score(val_label,y_predict,average='macro', zero_division=0)))
print("Recall : "+str(recall_score(val_label,y_predict,average='macro', zero_division=0)))
```

```
print("F1-Score : "+str(f1_score(val_label,y_predict,average='macro', zero_division=0)))
print("Classification Report")
print(classification_report(val_label,y_predict,zero_division=0))
print("Confusion Matrix")
print(confusion_matrix(val_label,y_predict))
#fig, ax = plt.subplots()
#plot_confusion_matrix(val_label, y_predict, ax=ax)


#neptune.log_metric('Training Accuracy', model.score(train_data,train_label))
#neptune.log_metric('Validation Accuracy', model.score(val_data,val_label))
#neptune.log_metric('Precision',precision_score(val_label,y_predict,average='macro', zero_division=0))
#neptune.log_metric('Recall', recall_score(val_label,y_predict,average='macro', zero_division=0))
#neptune.log_metric('F1-Score',f1_score(val_label,y_predict,average='macro', zero_division=0))

# Plot non-normalized confusion matrix
titles_options = [("Confusion matrix, without normalization", None),
                  ("Normalized confusion matrix", 'true')]
class_names = ["Check-In", "Canceled", "No-Show"]
for title, normalize in titles_options:
    disp = plot_confusion_matrix(model, val_data, val_label,
                                 display_labels=class_names,
                                 cmap=plt.cm.Blues,
                                 normalize=normalize)
    disp.ax_.set_title(title)

    print(title)
    print(disp.confusion_matrix)

#Feature Importance in XGBoost
print("Feature Importance")
print(model.feature_importances_) #use inbuilt class feature_importances of tree based classifiers
#plot graph of feature importances for better visualization
plt.figure(figsize=[10,10])
feat_importances = pd.Series(model.feature_importances_, index=train_data.columns)
feat_importances.nlargest(40).plot(kind='barh')

plt.show()
#print(feat_importances)
results=pd.DataFrame()
results['columns']=train_data.columns
results['importances'] = model.feature_importances_
results.sort_values(by='importances',ascending=False,inplace=True)

results[2:]
selected_features = results['columns'][2:].tolist()
print(selected_features)
```

```
"""##MLP classifier approach"""

from sklearn.neural_network import MLPClassifier
model = MLPClassifier(solver='adam',learning_rate = 'adaptive',learning_rate_init=0.01,activation=
'relu', alpha=1e-6, hidden_layer_sizes=(150, ), random_state=91,max_iter=400)
model.fit(train_data,train_label)
y_predict=model.predict(val_data)
print("Train accuracy : "+str(model.score(train_data,train_label)))
print("Validation accuracy : "+str(model.score(val_data,val_label)))
print("Precision : "+str(precision_score(val_label,y_predict,average='macro', zero_division=0)))
print("Recall : "+str(recall_score(val_label,y_predict,average='macro', zero_division=0)))
print("F1-Score : "+str(f1_score(val_label,y_predict,average='macro', zero_division=0)))
print("Classification Report")
print(classification_report(val_label,y_predict,zero_division=0))
print("Confusion Matrix")
print(confusion_matrix(val_label,y_predict))
#fig, ax = plt.subplots()
#plot_confusion_matrix(val_label, y_predict, ax=ax)


#neptune.log_metric('Training Accuracy', model.score(train_data,train_label))
#neptune.log_metric('Validation Accuracy', model.score(val_data,val_label))
#neptune.log_metric('Precision',precision_score(val_label,y_predict,average='macro', zero_division=0))
#neptune.log_metric('Recall', recall_score(val_label,y_predict,average='macro', zero_division=0))
#neptune.log_metric('F1-Score',f1_score(val_label,y_predict,average='macro', zero_division=0))

"""##Random Forest approach"""

#model = RandomForestClassifier(max_depth=7,max_features=10,n_estimators=75, class_weight =
'balanced' )
model = RandomForestClassifier(max_depth = 12, n_estimators = 115, class_weight = 'balanced',
random_state = 39  )
model.fit(train_data,train_label)
y_predict=model.predict(val_data)
print("Train accuracy : "+str(model.score(train_data,train_label)))
print("Validation accuracy : "+str(model.score(val_data,val_label)))
print("Precision : "+str(precision_score(val_label,y_predict,average='macro', zero_division=0)))
print("Recall : "+str(recall_score(val_label,y_predict,average='macro', zero_division=0)))
print("F1-Score : "+str(f1_score(val_label,y_predict,average='macro', zero_division=0)))
print("Classification Report")
print(classification_report(val_label,y_predict,zero_division=0))
print("Confusion Matrix")
print(confusion_matrix(val_label,y_predict))
#fig, ax = plt.subplots()
#plot_confusion_matrix(val_label, y_predict, ax=ax)


#neptune.log_metric('Training Accuracy', model.score(train_data,train_label))
```

```
#neptune.log_metric('Validation Accuracy', model.score(val_data,val_label))
#neptune.log_metric('Precision',precision_score(val_label,y_predict,average='macro', zero_division=0))
#neptune.log_metric('Recall', recall_score(val_label,y_predict,average='macro', zero_division=0))
#neptune.log_metric('F1-Score',f1_score(val_label,y_predict,average='macro', zero_division=0))

# Plot non-normalized confusion matrix
titles_options = [("Confusion matrix, without normalization", None),
                  ("Normalized confusion matrix", 'true')]
class_names = ["Check-In", "Canceled", "No-Show"]
for title, normalize in titles_options:
    disp = plot_confusion_matrix(model, val_data, val_label,
                                 display_labels=class_names,
                                 cmap=plt.cm.Blues,
                                 normalize=normalize)
    disp.ax_.set_title(title)

    print(title)
    print(disp.confusion_matrix)

#Feature Importance in Random Forest
print("Feature Importance")
print(model.feature_importances_) #use inbuilt class feature_importances of tree based classifiers
#plot graph of feature importances for better visualization
plt.figure(figsize=[10,10])
feat_importances = pd.Series(model.feature_importances_, index=train_data.columns)
feat_importances.nlargest(40).plot(kind='barh')

plt.show()
#print(feat_importances)
results=pd.DataFrame()
results['columns']=train_data.columns
results['importances'] = model.feature_importances_
results.sort_values(by='importances',ascending=False,inplace=True)

results[:30]
selected_features = results['columns'][:20].tolist()
print(selected_features)

"""##KNN approach"""

model=KNeighborsClassifier(n_neighbors=3,algorithm='auto',weights='distance')
model.fit(train_data,train_label)
y_predict=model.predict(val_data)
print("Train accuracy : "+str(model.score(train_data,train_label)))
print("Validation accuracy : "+str(model.score(val_data,val_label)))
print("Precision : "+str(precision_score(val_label,y_predict,average='macro', zero_division=0)))
print("Recall : "+str(recall_score(val_label,y_predict,average='macro', zero_division=0)))
print("F1-Score : "+str(f1_score(val_label,y_predict,average='macro', zero_division=0)))
```

```
print("Classification Report")
print(classification_report(val_label,y_predict,zero_division=0))
print("Confusion Matrix")
print(confusion_matrix(val_label,y_predict))
#fig, ax = plt.subplots()
#plot_confusion_matrix(val_label, y_predict, ax=ax)


#neptune.log_metric('Training Accuracy', model.score(train_data,train_label))
#neptune.log_metric('Validation Accuracy', model.score(val_data,val_label))
#neptune.log_metric('Precision',precision_score(val_label,y_predict,average='macro', zero_division=0))
#neptune.log_metric('Recall', recall_score(val_label,y_predict,average='macro', zero_division=0))
#neptune.log_metric('F1-Score',f1_score(val_label,y_predict,average='macro', zero_division=0))

# Plot non-normalized confusion matrix
titles_options = [("Confusion matrix, without normalization", None),
                  ("Normalized confusion matrix", 'true')]
class_names = ["Check-In", "Canceled", "No-Show"]
for title, normalize in titles_options:
    disp = plot_confusion_matrix(model, val_data, val_label,
                                 display_labels=class_names,
                                 cmap=plt.cm.Blues,
                                 normalize=normalize)
    disp.ax_.set_title(title)

    print(title)
    print(disp.confusion_matrix)

"""##Ensemble - extra tree classifier approach"""

from sklearn.ensemble import ExtraTreesClassifier
model = ExtraTreesClassifier(max_depth=12,n_estimators=100, class_weight = 'balanced')
model.fit(train_data, train_label)
y_predict= model.predict(val_data)
print("Train accuracy : "+str(model.score(train_data,train_label)))
print("Validation accuracy : "+str(model.score(val_data,val_label)))
print("Precision : "+str(precision_score(val_label,y_predict,average='macro', zero_division=0)))
print("Recall : "+str(recall_score(val_label,y_predict,average='macro', zero_division=0)))
print("F1-Score : "+str(f1_score(val_label,y_predict,average='macro', zero_division=0)))
print("Classification Report")
print(classification_report(val_label,y_predict,zero_division=0))
print("Confusion Matrix")
print(confusion_matrix(val_label,y_predict))
#fig, ax = plt.subplots()
#plot_confusion_matrix(val_label, y_predict, ax=ax)


#neptune.log_metric('Training Accuracy', model.score(train_data,train_label))
```

```
#neptune.log_metric('Validation Accuracy', model.score(val_data,val_label))
#neptune.log_metric('Precision',precision_score(val_label,y_predict,average='macro', zero_division=0))
#neptune.log_metric('Recall', recall_score(val_label,y_predict,average='macro', zero_division=0))
#neptune.log_metric('F1-Score',f1_score(val_label,y_predict,average='macro', zero_division=0))

"""##Hypertuning the model"""

#Hypertuning Parameters for Accuracy F1score and AUC score
#max_depth
#learning_rate
#min_child_weight
#gamma 4.63
#colsample_bytree
#scale_pos_weight
#subsample
#reg_lambda
#x=np.linspace(3,25,num=23,dtype=int)
x=np.linspace(0,500,num=1001,dtype=int)
train_acc = []
val_acc = []
F = []
#clf = DecisionTreeClassifier(max_depth=50, class_weight = 'balanced')
#x = [True, False]
for i in x:
    print(i)
    #model=xgboost.XGBClassifier(base_estimator = clf, max_depth = 19, n_estimators = 15, objective =
'multi:softmax', gamma = 4.5, learning_rate = i, reg_lambda = 3.4).fit(train_data,train_label)
#hypertuned model
    #model = DecisionTreeClassifier( max_depth = 20, class_weight = 'balanced', max_features = 'log2',
random_state = 8  ).fit(train_data,train_label)
    #model = RandomForestClassifier(max_depth = 12, n_estimators = 115, class_weight = 'balanced',
random_state = 39  ).fit(train_data,train_label)  # 115
    model = DecisionTreeClassifier(max_depth = None, min_samples_split=i, class_weight = 'balanced',
max_features = 'log2', random_state = 29  ).fit(train_data,train_label)
    y_pred= model.predict(val_data)
    f=f1_score(val_label,y_pred ,average='macro', zero_division=0)
    F.append(f)
    #auc=roc_auc_score(val_label,y_pred,average='macro')
    #AUC.append(auc)
    train_acc.append(model.score(train_data,train_label))
    val_acc.append(model.score(val_data,val_label))
    if f > 0.3697:
      print("improvement")
 #ploting hypertuning results

import matplotlib.pyplot as plt
#plt.plot(x,AUC)
#plt.title('AUC Score')
```

```
#plt.ylabel('Auc')
#plt.xlabel('Parameters')
#plt.show()
plt.plot(x,F)
plt.title('F1 Score')
plt.ylabel('F1')
plt.xlabel('Parameters')
plt.show()
plt.plot(x,train_acc)
plt.title('Train accuracy')
plt.ylabel('Acc')
plt.xlabel('Parameters')
plt.show()
plt.plot(x,val_acc)
plt.title('Validation Accuracy')
plt.ylabel('Acc')
plt.xlabel('Parameters')
plt.show()

print("Maximum Training Acc : "+str(max(train_acc)))
print(x[train_acc.index(max(train_acc))])

print("Maximum Validation Acc : "+str(max(val_acc)))
print(x[val_acc.index(max(val_acc))])

print("Maximum F1 Score : "+str(max(F)))
print(x[F.index(max(F))])

import matplotlib.pyplot as plt
x=x[:len(F)]
#plt.plot(x,AUC)
#plt.title('AUC Score')
#plt.ylabel('Auc')
#plt.xlabel('Parameters')
#plt.show()
plt.plot(x,F)
plt.title('F1 Score')
plt.ylabel('F1')
plt.xlabel('Parameters')
plt.show()
plt.plot(x,train_acc)
plt.title('Train accuracy')
plt.ylabel('Acc')
plt.xlabel('Parameters')
plt.show()
plt.plot(x,val_acc)
plt.title('Validation Accuracy')
plt.ylabel('Acc')
```

```
plt.xlabel('Parameters')
plt.show()

print("Maximum Training Acc : "+str(max(train_acc)))
print(x[train_acc.index(max(train_acc))])

print("Maximum Validation Acc : "+str(max(val_acc)))
print(x[val_acc.index(max(val_acc))])

print("Maximum F1 Score : "+str(max(F)))
print(x[F.index(max(F))])

"""#Prediction For submission"""

y_predict_2= model.predict_proba(test_data)
y_predict_2

test_2 = test_data['client_code']
test_2 = test_2.tolist()
y_predict_3 = []
for i in test_3:
  ind = test_2.index(i)
  y_predict_3.append(y_predict_2[ind])

print(len(test_3))
print(len(y_predict_3))

y_predict_3=pd.DataFrame(y_predict_3,columns=['probability_of_cross sell'] )
y_predict_3

test = pd.concat([test, y_predict_3],axis=1)
test.head()

test.to_csv('submission_Wanderers.csv',index=False)
```