Rachid LAMJOUN
AI & ML Master Program

Income Qualification

(Machine Learning: Project3)

Writeup

We have the subject of identifying the level of income qualification necessary for families in Latin America. The data to be studied correspond to the characteristics of Costa Rican households and observable domestic attributes of a family such as the material of their walls and ceilings etc.

We have the task to predict four qualification levels 1,2,3 and 4 from the poorest to the least poor.

Throughout our search for the best **model,** we will answer the following questions:

1. Identify the output variable.
2. Understand the type of data.
3. Check if there are any biases in your dataset.
4. Check whether all members of the house have the same poverty level.
5. Check if there is a house without a family head.
6. Set poverty level of the members and the head of the house within a family.
7. Count how many null values are existing in columns.
8. Remove null value rows of the target variable.
9. Predict the accuracy using random forest classifier.
10. Check the accuracy using random forest with cross validation.

We encountered missing data and following our analysis some data were completed while others were simply deleted. This was done in the preprocessing phase, see the preprocessing paragraph for more details.

There were two topics to deal with, the first one is the columns that contain null values :

```
X=df.isnull().sum().sort_values(ascending=False)
df_X=pd.DataFrame(X,columns=['nbre_of_null_values'])
df_X[df_X['nbre_of_null_values']>0]
```

	nbre_of_null_values
rez_esc	7928
v18q1	7342
v2a1	6860
SQBmeaned	5
meaneduc	5

For the columns ‘SBmeaned’ et ‘meaneduc’, there are **few rows with nulls,** we prefer to delete them.

We could find some logic in replacing the missing values of the 'rez_esq' column with 0, but deleting the column gives good results, and this is the option we have chosen.

Case of the column v18q1 (the number of tablets household owns). When v18q1 is nan, v18q (owns a tablet) is 0 and vice versa. Hence, to set v18q1=0 if it is nan, is compatible with the fact v18q=0 for all family members involved necessarily 0 for the household.

For v2a1 (Monthly rent payment), we can’t say for null values, v2a1=0 or not. However, one can imagine that the owners who have finished paying for their house le monthly rent payment can be 0 (v2a1=0). In other words, if tipovivi1 (own and fully paid house) =1, the v2a1=0. In the other cases (tipovivi1=0) the median is taken for v2a1.

The second to deal with, was object type columns, these are columns with alphanumeric values or a mix of numeric and string values. The columns are as follows:

```
df.select_dtypes('object').columns
```

```
Index(['Id', 'idhogar', 'dependency', 'edjefe', 'edjefa'],
dtype='object')
```

For the columns: 'dependency', 'edjefe' and 'edjefa' there were only two values which posed a problem ‘yes’ and ‘no’ which had to be replaced by numerical values.

The 'Id' column has been removed as a feature since it does not carry useful information for the ‘Target’ column.

For the column 'idhogar' it seems to us, that it carries the information of belonging to the same household, it has been coded in numerical.

Data upgrade work was initiated by the preprocessing() function, details of which can be found below.

For the construction part of the model, we first tested several classifiers. We used the test_classifierModels_list() function which takes as a parameter the list of classifiers that we want to test.

Specific pipelines for each algo have been implemented and the results have been evaluated by the fit_and_evaluate_model() function which will be behind all our model train operations and evaluations.

Classifiers tested are:

- **RandomForestClassifier**
- **AdaBoost**
- **SVM**
- **KNN**
- **DecisionTreeClassifier**

which gave the following results.

		precision		recall		f1-score
AdaBoost		0.39		0.37		0.36
SVM		0.49		0.34		0.33
KNN		0.61		0.60		0.60
DecisionTreeClassifier		0.83		0.83		0.83
RandomForestClassifier		0.86		0.83		0.84

We can clearly see, **DecisionTreeClassifie** and **RandomForestClassifier** which stand out.

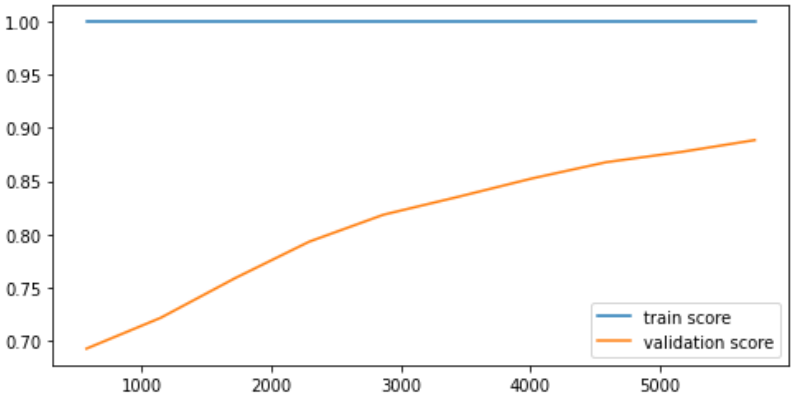
We focused our study on RandomForestClassifier model.

The **RandomForestClassifier** scores that we have just seen were obtained by applying the PolynomialFeatures and SelectKBest preprocessors.

Without using specific preprocessors and with random_state=0 the **RandomForestClassifier** obtained are much better:

random_state=0		precision		recall		f1-score
RandomForestClassifier		0.94		0.85		0.89

here we look by the learning curve to show the accuracy score evolution:



- The first optimization is obtained by filtering features. The **0.005** threshold is applied to feature_importances. The result is:

filtering of feature_importances by threshold 0.005		**precision**		**recall**		**f1-score**
RandomForestClassifier		0.95		0.90		0.92

We specify that the model was trained only by **64 features instead of 140**

- The second optimization consists in studying the correlation between features and by applying the threshold **0.85**, **the number of features goes from 64 to 48**. The result is the same (we lower the number of features without lowering performance):

filtering of feature_importances by threshold 0.005 and using 0.85 threshold for correlation between features		**precision**		**recall**		**f1-score**
RandomForestClassifier		0.95		0.90		0.92

- The third optimization consists in balancing the classes of the dataset, which, as we have seen, present a bias Q3. We use the SMOTE approach. The result is: we lose a point for the precision, on the other hand, we gain two points for the recall and one point for the f1-score.

filtering of feature_importances by threshold 0.005, using 0.85 threshold for correlation between features and enriching the dataset by SMOTE		**precision**		**recall**		**f1-score**
RandomForestClassifier		0.94		0.92		0.93

- For the fourth optimization we used **RandomizedSearchCV** function. Obviously, we put ourselves in the conditions of the previous optimizations to push the improvement of the metrics as far as possible. We recover the point of precision that we lost with the previous optimization.

filtering of feature_importances by threshold 0.005, using 0.85 threshold for correlation between features, enriching the dataset by SMOTE, and use RandomizedSearchCV		**precision**		**recall**		**f1-score**
RandomForestClassifier		**0.95**		**0.92**		**0.93**

DETAILS

Data Analysis

Loading Data

We start by loading the data**:**

```
import pandas as pd
# Loading train data
df_init = pd.read_csv('/content/train.csv')
df_init_test = pd.read_csv('/content/test.csv')
#
df=df_init.copy()
df_test=df_init_test.copy()
# Top 5 records
df.head()
```

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	SQBescolari
0	ID_279628684	190000.0	0	3	0	1	1	0	NaN	0	...	100
1	ID_f29eb3ddd	135000.0	0	4	0	1	1	1	1.0	0	...	144
2	ID_68de51c94	NaN	0	8	0	1	1	0	NaN	0	...	121
3	ID_d671db89c	180000.0	0	5	0	1	1	1	1.0	0	...	81
4	ID_d56d6f5f5	180000.0	0	5	0	1	1	1	1.0	0	...	121

5 rows × 143 columns

```
df.shape
```

```
(9557, 143)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9557 entries, 0 to 9556
Columns: 143 entries, Id to Target
dtypes: float64(8), int64(130), object(5)

memory usage: 10.4+ MB
```

Q1. Identify the output variable.

The out variable is **Target** in file train.csv but not in test.csv

We verify that all columns of **test.csv** are the same that **train.csv** only Target column.

```
len(df.columns),len(df_test.columns)
```

```
(143, 142)
```

```
[col for col in df.columns if col not in df_test.columns]
```

```
['Target']
```

For a supervised approach, the test.csv file will not be of significant use to us.

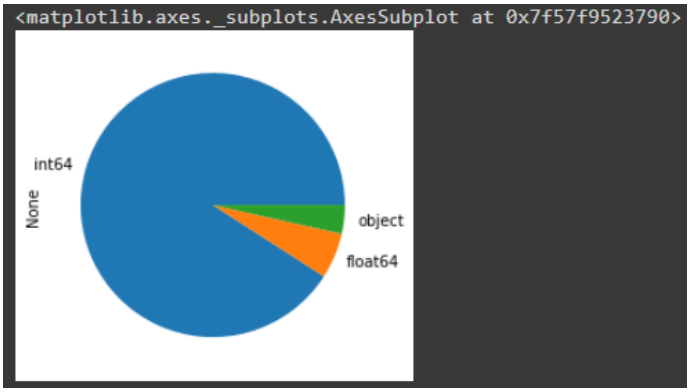
Q2. Understand the type of data

```
df.dtypes.value_counts()
```

```
int64      130
float64      8
object       5
dtype: int64
```

```
df.dtypes.value_counts().plot.pie()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f57f9523790>
```



We try to understand columns with object type:

```
df.select_dtypes('object').columns
```

```
Index(['Id', 'idhogar', 'dependency', 'edjefe', 'edjefa'],
dtype='object')
```

For each column of type object, we give the number of values:

```
# list of columns with count of its values by type of columns
def get_value_counts(i_df,i_type):
  lst_nbre_val_col=[]
  cmpt=0
  for col in i_df.select_dtypes(i_type).columns:
    cmpt=i_df[col].value_counts().count()
    lst_nbre_val_col.append((col,cmpt))
  #
  return lst_nbre_val_col

get_value_counts(df,'object')
```

```
[('Id', 9557),
 ('idhogar', 2988),
 ('dependency', 31),
 ('edjefe', 22),
 ('edjefa', 22)]
```

Seeing a few rows allows us to have an idea about the types of column values.

```
df.select_dtypes('object').head()
```

	Id	idhogar	dependency	edjefe	edjefa
0	ID_279628684	21eb7fcc1	no	10	no
1	ID_f29eb3ddd	0e5d7a658	8	12	no
2	ID_68de51c94	2c7317ea8	8	no	11
3	ID_d671db89c	2b58d945f	yes	11	no
4	ID_d56d6f5f5	2b58d945f	yes	11	no

the columns **Id** and **idhogar** look like **alphanumeric** values and we look more precisely at the values of the other low cardinality columns: **dependency**, **edjefe** and **edjefa.**

```
# Get val no num from dependency, edjefe and edjefa columns
def get_lst_no_num(i_df,i_col):
  l_X=i_df[i_col]
  ll=l_X.unique()
  l_lst_no_num=[]
  # print(ll)
  for i in ll:
    try:
      float(i)
    except:
      l_lst_no_num.append(i)
  #
  return l_lst_no_num
#
get_lst_no_num
```

By this function we found only the values: ['no', 'yes'] for the three columns.

```
print('dependency-->',get_lst_no_num(df,'dependency'))
print('edjefe-->',get_lst_no_num(df,'edjefe'))
print('edjefa-->',get_lst_no_num(df,'edjefa'))
```

```
dependency--> ['no', 'yes']
edjefe--> ['no', 'yes']
edjefa--> ['no', 'yes']
```

we now look at the other types of columns. Also, by applying the function:
get_value_counts(i_df, i_type)
we find that few columns have a number of values that exceed **10**, and in any case, for the two types **float64** and **int64** the values do not exceed **156** as shown below.

```
# type= float64
lst_t=get_value_counts(df,'float64')
[i for i in lst_t if i[1] > 10]
```

```
[('v2a1', 156),
 ('meaneduc', 154),
 ('overcrowding', 38),
 ('qmobilephone', 11),
 ('age', 97),
 ('SQBescolari', 22),
 ('SQBage', 97),
 ('SQBhogar_total', 13),
 ('SQBedjefe', 22),
 ('SQBovercrowding', 38),
 ('SQBdependency', 29),
 ('SQBmeaned', 154),
 ('agesq', 97)]
```

```
# type= int64
lst_t=get_value_counts(df,'int64')
[i for i in lst_t if i[1] > 10]
```

```
[('rooms', 11),
 ('r4t2', 11),
 ('r4t3', 13),
 ('tamhog', 13),
 ('tamviv', 14),
 ('escolari', 22),
 ('hhsize', 13),
 ('hogar_total', 13)]
```

Q3. Check if there are any biases in your dataset.

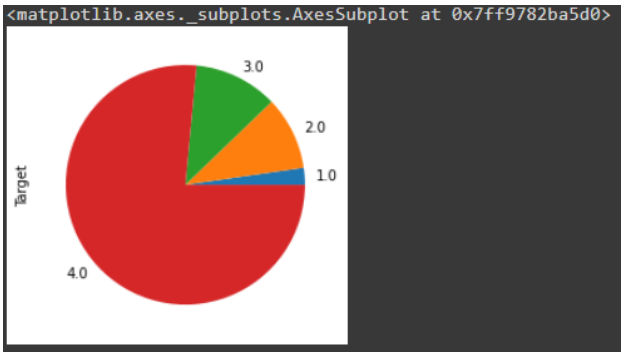
We look at the size of each class.

```
df.groupby('Target')['Target'].sum()
```

```
Target
1.0      704.0
2.0     3050.0
3.0     3486.0
4.0    23560.0
Name: Target, dtype: float64
```

```
df.groupby('Target')['Target'].sum().plot.pie()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff9782ba5d0>
```



The other classes are poorly represented compared to class 4, and the class one is, by far, poorly represented compared to the others. So, we have a dataset bias.

Q4. Check whether all members of the house have the same poverty level.

if this were the case, we would have len1 and len2 equal but unfortunately this is not the case according to the following.

```
len1=len(df['idhogar'].unique())
print("len1=",len1)
```

```
len1= 2988
```

```
len2=len(df.groupby(['idhogar','Target'])['Target'].unique())
print("len2=",len2)
```

```
len2= 3074
```

Q5. Check if there is a house without a family head.

```
# Family without the head
X=df[df['parentesco1']==1]
len(X['idhogar'].unique()),len(df['idhogar'].unique())
```

```
(2973, 2988)
```

We see that, 15 households without a family head

Q6. Set poverty level of the members and the head of the house within a family.

We will rely on this function which lists by default the families with different Targets from its members.

```
# list families with multiple Target members
# Update the target of family members by that of the header (set #
                                                  i_update_target=True)
def list_family_no_unique_target(i_df,i_update_target=False):
  list_h=i_df[i_df['parentesco1']==1]
  #len(list_h)
  ll=list(list_h['idhogar'])
  ll2=[]
  for idh in ll:
    X_target=list_h[list_h['idhogar']==idh].iloc[:]['Target']
    target=X_target.values[0]
    XX=set(i_df[i_df['idhogar']==idh]['Target'])
    if len(XX) > 1:
      ll2.append((idh,len(XX),target))
      if i_update_target:
        df.loc[df.idhogar==idh,'Target']=target
  return(ll2)
```

By passing the **i_update_target** parameter as True, the updates of the Targets are put in place so that all the members of the family have the Target of the head of the house within family.

In the first time we check the existence of such family:

```
# list families with multiple Target members
len(list_family_no_unique_target(df))
```

```
85
```

We then update the Targets

```
# Update the Target of family members by that of the header
len(list_family_no_unique_target(df,True))
```

```
85
```

At the end we check if there are families left with members of different Targets

```
# check if there are families left with members of different Targets
len(list_family_no_unique_target(df))
```

```
0
```

Q7. Count how many null values are existing in columns.

```
X=df.isnull().sum().sort_values(ascending=False)
df_X=pd.DataFrame(X,columns=['nbre_of_null_values'])
df_X[df_X['nbre_of_null_values']>0]
```

	nbre_of_null_values
rez_esc	7928
v18q1	7342
v2a1	6860
SQBmeaned	5
meaneduc	5

There are only five columns with null values:

Rez_esc with 7928 null
V18q1 with 7342 null
V2a1 with 6860 null
SQBmeaned with 5 null
meanduc with 5 null

Q8. Remove null value rows of the target variable.

According to question Q7 the Target column does not appear in the list of columns with nulls.

Preprocessing

The goal is to find a solution for all nulls and ultimately have numeric columns.

```
# Encoding to 0,1 teh values 'yes' and 'no' for dependency, edjefe ,
#                                                              edjefa
# Encoding to numeric idhogar
def encodage(i_df):
    code = {'yes':1,'no':0}
    l_df=i_df.copy()
    # 'dependency', 'edjefe', 'edjefa']
    mapping={'yes':1,'no':0}

    l_df['dependency'] =l_df['dependency'].replace(code).
                                                  astype(np.float64)
```

```
    l_df['edjefe'] =l_df['edjefe'].replace(code).astype(np.float64)
    l_df['edjefa'] =l_df['edjefa'].replace(code).astype(np.float64)

    #
    # Encodage idhogar column
    set1=set(l_df['idhogar'])
    ll=list(set1)
    #
    l_mapping={ll[i]:i+1 for i in range(0, len(ll))}
    l_df['idhogar'] =l_df['idhogar'].replace(l_mapping).
                                                astype(np.float64)
    #idhogar
    return l_df
```

Case of v18q1 and v2a1 columns.

Case of v18q1

```
# we prove when v18q1 is null necessarily v18q is 0.
# So, each family member hasn't a tablet i.e. v18q=0
X=df[['idhogar','v18q1','v18q']]
X=X[X['v18q1'].isna()]
X['v18q'].unique()
```

```
array([0])
```

```
# we have v18q=0 ==> v18q1=nan
X=df[['idhogar','v18q1','v18q']]
X=X[X['v18q']==0]
X['v18q1'].unique()
```

```
array([nan])
```

Case of v2a1

```
X = df[df['v2a1'].isnull()]
X=X[df.columns]
for col in ['tipovivi1',  'Target']:
 print(col, '    ', X[col].unique())
```

```
tipovivi1      [1 0]
Target      [4 2 3 1]
```

```
X['tipovivi1'].value_counts()
```

```
1    5911
0     949
Name: tipovivi1, dtype: int64
```

```
# Don't forget to compile list_family_no_unique_target()
# in family information section
def feature_engineering(i_df):
  l_df=i_df
  #
  # Update the target of family members by that of the header
  list_family_no_unique_target(l_df,True)
  #
  l_df.loc[l_df['tipovivi1']==1,'v2a1']=0
  l_df['v2a1'].replace([np.nan], l_df['v2a1'].median(), inplace=True)
  #
  l_df['v18q1'].fillna(value=0, inplace=True)
  return l_df
```

```
# prepa imputation()
# drop rows for meaneduc and SQBmeaned
def preprocess_del_na(i_df):
  #
  l_X=i_df.loc[:,'meaneduc']
  l_X=pd.DataFrame(l_X,columns=['meaneduc'])
  l_df=i_df[l_X['meaneduc'].isna() == False]
  #
  l_X=l_df.loc[:,'SQBmeaned']
  l_X=pd.DataFrame(l_X,columns=['SQBmeaned'])
  l_X=l_df[l_X['SQBmeaned'].isna() == False]
  #

  # print('df.shape=',l_df.shape)
  return l_X
```

```
# drop columns rez_esc and Id
def imputation(i_df):
  l_df=preprocess_del_na(i_df) # del row with nan
  #
  cols=['rez_esc', 'Id']
  l_df.drop(columns = cols,axis=1,inplace=True)
  #
  return  l_df
```

Our preprocessing function:

preprocessing(i_df,i_col_target):

We pass as parameters the dataframe of the data and the column 'Target' and which gives as output a dataframe (l_X) without null values and only numeric values, and a Series (l_y) which corresponds to the column 'Target'.

```
# Encodage, imputation and engineering for features
# df treated without column Target
contains only the column Target
def preprocessing(i_df,i_col_target):

    l_df=i_df
    l_df = feature_engineering(l_df)
    l_df=encodage(l_df)
    l_df = imputation(l_df)


    l_X = l_df.drop(i_col_target, axis=1)
    l_y = l_df[i_col_target]

    return l_X,l_y
```

Reinit Preprocess and Split Data

```
# Reinit data to facilitate testing
def reinit_preprocess_split():
  # Reinit
  l_df=reinit_input_df()
  print('df.shape=',df.shape)
  #
  # Preprocess
  (X,y)=preprocessing(l_df,'Target')
  X.shape,y.shape
  #
  #***************** Checking *********************#
  #
  # Null operation
  Z=X.isnull().sum().sort_values(ascending=False) > 0
  ll=[col for col in Z.index if Z.loc[col]==True]
  print('list columns with Null:',ll)
  #
  print('list_family_no_unique_target()--
>',list_family_no_unique_target(df,False)) # no update only checking th
e existance
  #
  print('No numeric in dependency-->',get_lst_no_num(X,'dependency'))
  print('No numeric in edjefe-->',get_lst_no_num(X,'edjefe'))
  print('No numeric in edjefa-->',get_lst_no_num(X,'edjefa'))
  #
  print('No numeric in idhogar-->',get_lst_no_num(X,'idhogar'))
```

```
  #***************** end Checking *********************#
  #
  # Split data
  l_X_train,l_X_test,l_y_train,l_y_test=train_test_split(X,y,test_size=
0.2,random_state=1)
  print('X_train.shape, X_test.shape,y_train.shape,y_test.shape')
  print(l_X_train.shape, l_X_test.shape,l_y_train.shape,l_y_test.shape)
  return (l_df,(l_X_train,l_X_test,l_y_train,l_y_test))
```

```
(df,(X_train,X_test,y_train,y_test))=reinit_preprocess_split()
```

```
df.shape= (9557, 143)
list columns with Null: []
list_family_no_unique_target()--> []
No numeric in dependency--> []
No numeric in edjefe--> []
No numeric in edjefa--> []
No numeric in idhogar--> []
X_train.shape, X_test.shape,y_train.shape,y_test.shape
(7641, 140) (1911, 140) (7641,) (1911,)
```

Evaluate different models

First, we want to point out that **the tested models are**: RandomForestClassifier, AdaBoost, SVM, KNN and DecisionTreeClassifier

For information we need these imports:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.metrics import confusion_matrix, classification_report
```

All our evaluations are carried out by this function:

fit_and_evaluate_model()

```
############################
### Evaluation function
############################
# Used to Evaluate models
```

```
def fit_and_evaluate_model(i_model,i_X_train,i_X_test,i_y_train,i_y_tes
t):
  #
  l_model=i_model
  l_model.fit(i_X_train, i_y_train)
  l_ypred = l_model.predict(i_X_test)

  print(confusion_matrix(i_y_test, l_ypred))
  print(classification_report(i_y_test, l_ypred))

  return l_model
```

The **important** function of this paragraph is the following and which based on the pipelines built for our classifiers to be tested.

test_classifierModels_list()

```
########################################
### test_classifierModels_list function
########################################
def test_classifierModels_list(i_classifierModels_list,i_X_train,i_X_te
st,i_y_train,i_y_test):
  l_dico_name_model={}
  preprocessor = make_pipeline(PolynomialFeatures(2, include_bias=False
), SelectKBest(f_classif, k=10))
  RandomForest = make_pipeline(preprocessor, RandomForestClassifier(ran
dom_state=0))
  AdaBoost = make_pipeline(preprocessor, AdaBoostClassifier(random_stat
e=0))
  SVM = make_pipeline(preprocessor, StandardScaler(), SVC(random_state=
0))
  KNN = make_pipeline(preprocessor, StandardScaler(), KNeighborsClassif
ier())
  DTC = make_pipeline(preprocessor, StandardScaler(), DecisionTreeClass
ifier())
  dict_of_models = {'RandomForestClassifier': RandomForest,
                    'AdaBoost' : AdaBoost,
                    'SVM': SVM,
                    'KNN': KNN,
                    'DecisionTreeClassifier':DTC
                   }
  for name, model in dict_of_models.items():
    if name in i_classifierModels_list:
      print('============')
      print(name)
      print('============')
```

```
        l_model= fit_and_evaluate_model(model,i_X_train,i_X_test,i_y_trai
n,i_y_test)
        l_dico_name_model[name]=l_model
        #
    return l_dico_name_model
```

The call to this function is done as by specifying the list of models that we want to test:

```
# list classifiers
lst_algo=['AdaBoost','SVM','KNN','DecisionTreeClassifier','RandomForest
Classifier']
#
dico_name_model=test_classifierModels_list(lst_algo,X_train,X_test,y_tr
ain,y_test)
dico_name_model
```

The test clearly showed that **RandomForestClassifier** and **DecisionTreeClassifier** give good results while the others are not good (we look at the averages for the three metrics: accuracy, recall and f1):

```
RandomForestClassifier
============
[[ 119   13    3   14]
 [   8  254   11   33]
 [   3   22  166   39]
 [   3   15   30 1178]]
              precision    recall  f1-score   support

           1       0.89      0.80      0.84       149
           2       0.84      0.83      0.83       306
           3       0.79      0.72      0.75       230
           4       0.93      0.96      0.95      1226

    accuracy                           0.90      1911
   macro avg       0.86      0.83      0.84      1911
weighted avg       0.90      0.90      0.90      1911
```

```
============
DecisionTreeClassifier
============
[[ 125   12    6    6]
 [  15  257   12   22]
 [   5   31  165   29]
 [   8   31   34 1153]]
              precision    recall  f1-score   support

           1       0.82      0.84      0.83       149
           2       0.78      0.84      0.81       306
           3       0.76      0.72      0.74       230
           4       0.95      0.94      0.95      1226

    accuracy                           0.89      1911
   macro avg       0.83      0.83      0.83      1911
weighted avg       0.89      0.89      0.89      1911
```

```
============
AdaBoost
============
[[  26   51    2   70]
 [  21  112    3  170]
 [   3   44    0  183]
 [  16   57    2 1151]]
              precision    recall  f1-score   support

           1       0.39      0.17      0.24       149
           2       0.42      0.37      0.39       306
           3       0.00      0.00      0.00       230
           4       0.73      0.94      0.82      1226

    accuracy                           0.67      1911
   macro avg       0.39      0.37      0.36      1911
weighted avg       0.57      0.67      0.61      1911
```

```
============
SVM
============
[[  11   62    0   76]
 [   2   93    0  211]
 [   0   34    0  196]
 [   0   38    0 1188]]
              precision    recall  f1-score   support

           1       0.85      0.07      0.14       149
           2       0.41      0.30      0.35       306
           3       0.00      0.00      0.00       230
           4       0.71      0.97      0.82      1226

    accuracy                           0.68      1911
   macro avg       0.49      0.34      0.33      1911
weighted avg       0.59      0.68      0.59      1911
```

```
============
KNN
============
[[  69   25   19   36]
 [  21  193   21   71]
 [  16   40   96   78]
 [  33   63   55 1075]]
              precision    recall  f1-score   support

           1       0.50      0.46      0.48       149
           2       0.60      0.63      0.62       306
           3       0.50      0.42      0.46       230
           4       0.85      0.88      0.86      1226

    accuracy                           0.75      1911
   macro avg       0.61      0.60      0.60      1911
weighted avg       0.74      0.75      0.75      1911
```

RandomForestClassifier Model

In this paragraph we build a Random Forest Classifier model that we will evaluate it, which after we will compare it with other types of models, and will also, be the basis for building our optimization path for the final target model.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import learning_curve
```

```
################################
### function show_learning_curve
################################
  # evolution of accuracy
  # comparing X_train and X_test accuracy
def show_learning_curve(i_model, i_X_train, i_y_train,i_scoring='accura
cy'):
  N, train_score, val_score = learning_curve(i_model, i_X_train,
                                                          i_y_train,
                                             cv=4, scoring=i_scoring,
                                               train_sizes=np.linspace(0.1
, 1, 10))
  plt.figure(figsize=(8, 4))
  plt.plot(N, train_score.mean(axis=1), label='train score')
  plt.plot(N, val_score.mean(axis=1), label='validation score')
  plt.legend()
```

```
# Initilization
(df,(X_train,X_test,y_train,y_test))=reinit_preprocess_split()
```

Q.9 Predict the accuracy using random forest classifier.

we use the classifiers test function,
test_classifierModels_list function()
of the paragraph "Evaluate different models". We pass it, therefore, as an argument
['RandomForestClassifier']

```
# RandomForestClassifier model
# RandomForestClassifier using pipeline the previous paragraph
# (using preprocessors PolynomialFeatures()and SelectKBest())

dico_t=test_classifierModels_list(['RandomForestClassifier'],X_train,X_
test,y_train,y_test)
model_rfc=dico_t['RandomForestClassifier']
model_rfc
```

```
============
RandomForestClassifier
============
[[ 119   13    3   14]
 [   8  254   11   33]
 [   3   22  166   39]
 [   3   15   30 1178]]
              precision    recall  f1-score   support

           1       0.89      0.80      0.84       149
           2       0.84      0.83      0.83       306
           3       0.79      0.72      0.75       230
           4       0.93      0.96      0.95      1226

    accuracy                           0.90      1911
   macro avg       0.86      0.83      0.84      1911
weighted avg       0.90      0.90      0.90      1911

Pipeline(steps=[('pipeline',
                 Pipeline(steps=[('polynomialfeatures',
                                  PolynomialFeatures(include_bias=False)),
                                 ('selectkbest', SelectKBest())])),
                ('randomforestclassifier',
                 RandomForestClassifier(random_state=0))])
```

Q10. Check the accuracy using random forest with cross validation

```
#  Cross Validation by KFold
from sklearn.model_selection import KFold,cross_val_score
seed=7
kfold=KFold(n_splits=5,random_state=seed,shuffle=True)

model_rfc=RandomForestClassifier(n_estimators=100, random_state=42)
```

```
cv_model_rfc=cross_val_score(model_rfc,X_train,y_train,cv=kfold,scoring
='accuracy')
print(cv_model_rfc)
#
fit_and_evaluate_model(model_rfc,X_train,X_test,y_train,y_test)
```

```
[0.90189666 0.90903141 0.90510471 0.89267016 0.89136126]
[[ 120    9    0   20]
 [   5  259    2   40]
 [   0    8  173   49]
 [   0    4    7 1215]]
              precision    recall  f1-score   support

           1       0.96      0.81      0.88       149
           2       0.93      0.85      0.88       306
           3       0.95      0.75      0.84       230
           4       0.92      0.99      0.95      1226

    accuracy                           0.92      1911
   macro avg       0.94      0.85      0.89      1911
weighted avg       0.93      0.92      0.92      1911

RandomForestClassifier(random_state=42)
```

Optimization 1

```
#################################
### function show_importances_plot
#################################
# Only importances > i_threshold
def show_importances_plot(i_model, i_threshold,i_figsize=(11, 6)):
  l_feature_importances_df=pd.DataFrame({'importances': i_model.feature
_importances_, 'columns_': i_model.feature_names_in_})
  l_feature_importances_df.sort_values(by=['importances'], ascending=Tr
ue, inplace=True)
  feature_importances_df_t=l_feature_importances_df[l_feature_importanc
es_df['importances']> i_threshold]
  feature_importances_df_t.set_index('columns_',inplace=True)
  #
  feature_importances_df_t.plot(figsize=i_figsize,kind='barh')
  return l_feature_importances_df
```

```
#################################
### get_list_columns_by_threshold
#################################
def get_list_columns_by_threshold(i_feature_importances_df,i_threshold=
0.005):
```

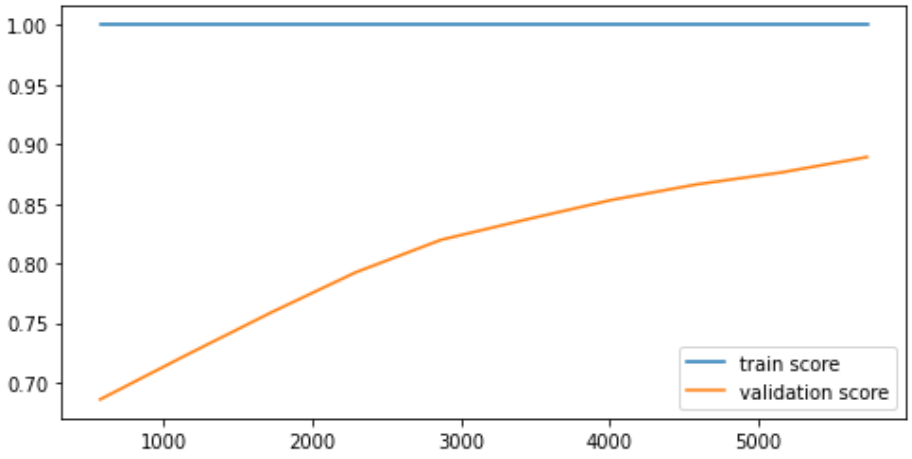
```
  l_feature_importances_df_t=i_feature_importances_df[i_feature_importa
nces_df.importances > i_threshold]
  return list(l_feature_importances_df_t.columns_)
#
```

```
# Initilization
(df,(X_train,X_test,y_train,y_test))=reinit_preprocess_split()
```

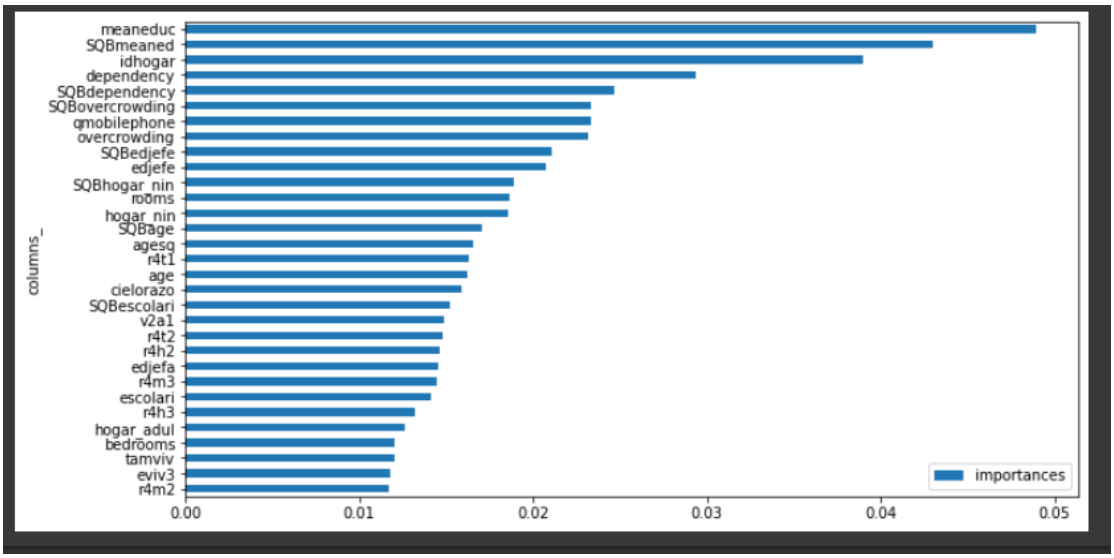
```
# RandomForestClassifier model without use the specific preprocessors
model_rfc = RandomForestClassifier(random_state=0)
fit_and_evaluate_model(model_rfc,X_train,X_test,y_train,y_test)
```

```
[[ 123    9    0   17]
 [   3  257    6   40]
 [   0    7  173   50]
 [   0    2    4 1220]]
              precision    recall  f1-score   support

           1       0.98      0.83      0.89       149
           2       0.93      0.84      0.88       306
           3       0.95      0.75      0.84       230
           4       0.92      1.00      0.96      1226

    accuracy                           0.93      1911
   macro avg       0.94      0.85      0.89      1911
weighted avg       0.93      0.93      0.93      1911

RandomForestClassifier(random_state=0)
```

```
# evolution of accuracy
# comparing X_train and X_test accuracy
show_learning_curve(model_rfc,X_train,y_train,'accuracy')
```



```
feature_importances_df=show_importances_plot(model_rfc,0.011)
```



filtering features by applying the 0.005 threshold to feature_importances:

```
list_cols_importances_0_005=get_list_columns_by_threshold
                                          (feature_importances_df,0.005)
len(list_cols_importances_0_005)
```

```
64
```

```
# Initilization
(df,(X_train,X_test,y_train,y_test))=reinit_preprocess_split()
```

```
X_train.shape,X_test.shape
```

```
((7641, 140), (1911, 140))
```

```
X_train=X_train[list_cols_importances_0_005]
#
X_test=X_test[list_cols_importances_0_005]
#
X_train.shape,X_test.shape
```

```
((7641, 64), (1911, 64))
```

```
# RandomForestClassifier model
```

```
model_rfc = RandomForestClassifier(random_state=0)
fit_and_evaluate_model(model_rfc,X_train,X_test,y_train,y_test)
```

```
[[ 129    9    0   11]
 [   5  272    4   25]
 [   0    7  192   31]
 [   0    3    3 1220]]
              precision    recall  f1-score   support

           1       0.96      0.87      0.91       149
           2       0.93      0.89      0.91       306
           3       0.96      0.83      0.90       230
           4       0.95      1.00      0.97      1226

    accuracy                           0.95      1911
   macro avg       0.95      0.90      0.92      1911
weighted avg       0.95      0.95      0.95      1911

RandomForestClassifier(random_state=0)
```

Optimization 2

```
# Initilization
(df,(X_train,X_test,y_train,y_test))=reinit_preprocess_split()
```

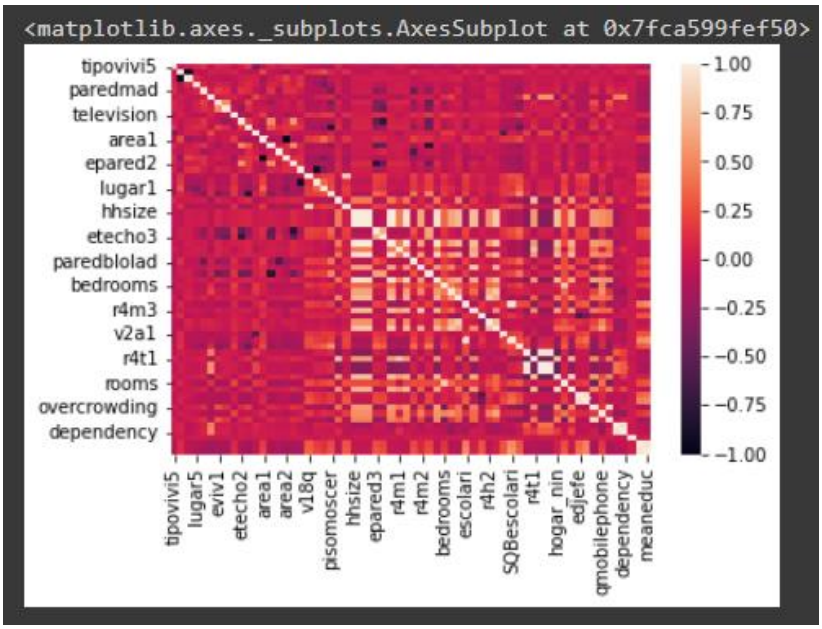
```
X_train=X_train[list_cols_importances_0_005]
#
X_test=X_test[list_cols_importances_0_005]
#
X_train.shape,X_test.shape
```

```
((7641, 66), (1911, 66))
```

```
# Create correlation matrix
corr_matrix = X_train.corr()
corr_matrix.shape
```

```
(64, 64)
```

```
sns.heatmap(corr_matrix)
```




```
# looking for features corresponding at the threshold 0.80
cols=corr_matrix.columns
l11=[]  # for help to filtring
l12=[]  # columns to keep after filtring
cmpt=0
for col1 in cols:
  for col2 in cols:
    l_val=corr_matrix.loc[col2,col1]
    if cmpt < 10:
      #print (abs(l_val))
      cmpt=cmpt+1
    if abs(l_val) > 0.85:
      if col1!=col2:
        if col1 not in l12:
          l11.append(col1)
        l12.append(col2)
len(l11),len(l12)
```

```
(16, 54)
```

```
# remove duplicates
l11_unique=list(set(l11))
l12_unique=list(set(l12))
len(l11_unique),len(l12_unique)
```

```
(11, 27)
```

```
# features to eliminate
lst_a_droper=[col for col in l12_unique if col not in l11 ]
```

```
len(lst_a_droper)
```

```
16
```

```
# list of features to keep for after
list_cols_above_0_80=[col for col in cols if col not in lst_a_droper]
len(list_cols_above_0_80)
```

```
48
```

filtering features by applying the 0.005 threshold to feature_importances and 0.85 threshold for correlation:

```
X_train=X_train[list_cols_above_0_80]
#
X_test=X_test[list_cols_above_0_80]
#
X_train.shape,X_test.shape
```

```
((7641, 48), (1911, 48))
```

```
# RandomForestClassifier model
model_rfc = RandomForestClassifier(random_state=0)
fit_and_evaluate_model(model_rfc,X_train,X_test,y_train,y_test)
```

```
[[ 131    8    0   10]
 [   5  271    8   22]
 [   0   12  194   24]
 [   0    1    6 1219]]
              precision    recall  f1-score   support

           1       0.96      0.88      0.92       149
           2       0.93      0.89      0.91       306
           3       0.93      0.84      0.89       230
           4       0.96      0.99      0.97      1226

    accuracy                           0.95      1911
   macro avg       0.95      0.90      0.92      1911
weighted avg       0.95      0.95      0.95      1911

RandomForestClassifier(random_state=0)
```

Optimization 3

```
# Initilization
(df,(X_train,X_test,y_train,y_test))=reinit_preprocess_split()
```

Balancing the classes of the dataset by SMOTE approach

```
###################################
### SMOTE() Operation
###################################

from enum import auto
#
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=20)
#'minority','auto'equivalent à 'not majority','not minority','all'
X_train_res, y_train_res = sm.fit_resample(X_train, y_train.ravel())

X_train_res.shape,y_train_res.shape
```

```
((19060, 140), (19060,))
```

```
sm.get_params()
```

```
{'k_neighbors': 5,
 'n_jobs': None,
 'random_state': 20,
 'sampling_strategy': 'auto'}
```

```
sum(y_train_res == 1),sum(y_train_res == 2),sum(y_train_res == 3),
                                              sum(y_train_res == 4)
```

```
# affectation
X_train,y_train=X_train_res, y_train_res
```

```
X_train.shape,y_train.shape
```

```
((19060, 140), (19060,))
```

```
#=============================================
# Case2 Optimization2
X_train=X_train[list_cols_above_0_80]
#
X_test=X_test[list_cols_above_0_80]
#
X_train.shape,X_test.shape
#=============================================
```

```
#
```

```
((19060, 48), (1911, 48))
```

```
# RandomForestClassifier model
model_rfc = RandomForestClassifier(random_state=0)
fit_and_evaluate_model(model_rfc,X_train,X_test,y_train,y_test)
```

```
 [[ 135    7    1    6]
  [   5  278    7   16]
  [   0   13  200   17]
  [   3    3    6 1214]]
               precision    recall  f1-score   support

            1       0.94      0.91      0.92       149
            2       0.92      0.91      0.92       306
            3       0.93      0.87      0.90       230
            4       0.97      0.99      0.98      1226

     accuracy                           0.96      1911
    macro avg       0.94      0.92      0.93      1911
 weighted avg       0.96      0.96      0.96      1911

 RandomForestClassifier(random_state=0)
```

Optimization 4

```
# Initilization
(df,(X_train,X_test,y_train,y_test))=reinit_preprocess_split()
```

```
# SMOTE
sm = SMOTE(random_state=20)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train.ravel())
X_train_res.shape,y_train_res.shape
```

```
((19060, 140), (19060,))
```

```
# here assignment
X_train,y_train=X_train_res, y_train_res
#
#============================================
# Apply the threshold 0_005 for features_imprtances
X_train=X_train[list_cols_importances_0_005]
#
X_test=X_test[list_cols_importances_0_005]
#
X_train.shape,X_test.shape
#============================================
#============================================
# Apply the threshold 0_85 for the correlation
```

```
X_train=X_train[list_cols_above_0_80]
#
X_test=X_test[list_cols_above_0_80]
#
X_train.shape,X_test.shape
#=============================================
```

```
((19060, 48), (1911, 48))
```

RandomizedSearchCV

```
# RandomizedSearchCV
from sklearn.model_selection import RandomizedSearchCV

#
rfc=RandomForestClassifier(bootstrap=True, class_weight=None, max_leaf_
nodes=None,
                                min_impurity_decrease=0.0,
                                    min_samples_leaf=1, min_samples_spl
it=2,
                                        min_weight_fraction_leaf=0.0, n
_estimators=0, n_jobs=2,
                                            oob_score=False, random_sta
te=0, verbose=0, warm_start=False)

tuned_parameters={'n_estimators': [200,300,400], 'max_features': ['auto
', 'sqrt', 'log2'],'max_depth': [32,64], 'criterion': ['gini', 'entropy
']}

#
CV_rfc=RandomizedSearchCV(cv=5, error_score='raise', estimator=rfc, par
am_distributions=tuned_parameters,n_jobs=2,
                                   pre_dispatch='2*n_jobs', refit=True,
 return_train_score='warn',
                                       scoring=None, verbose=0)

#
CV_rfc.fit(X_train, y_train)

CV_rfc.best_params_
#
fit_and_evaluate_model(CV_rfc,X_train,X_test,y_train,y_test)
```

```
[[ 135    8    0    6]
 [   5  276    8   17]
 [   0   13  201   16]
 [   0    3    7 1216]]
              precision    recall  f1-score   support

           1       0.96      0.91      0.93       149
           2       0.92      0.90      0.91       306
           3       0.93      0.87      0.90       230
           4       0.97      0.99      0.98      1226

    accuracy                           0.96      1911
   macro avg       0.95      0.92      0.93      1911
weighted avg       0.96      0.96      0.96      1911

RandomizedSearchCV(cv=5, error_score='raise',
                   estimator=RandomForestClassifier(n_estimators=0, n_jobs=2,
                                                    random_state=0),
                   n_jobs=2,
                   param_distributions={'criterion': ['gini', 'entropy'],
                                        'max_depth': [32, 64],
                                        'max_features': ['auto', 'sqrt',
                                                         'log2'],
                                        'n_estimators': [200, 300, 400]},
                   return_train_score='warn')
```

```
CV_rfc.best_params_
```

```
{'criterion': 'gini',
 'max_depth': 32,
 'max_features': 'log2',
 'n_estimators': 400}
```