Rachid LAMJOUN
PG AI Program

Lending Club Loan Data Analysis

PG AI – Deep Learning with Tensorflow and Keras Project:

Project1: Lending Club Loan Data Analysis

Writeup

We have the subject of building a deep learning model to predict the chance of default for future loans.

The model will be based on a dataset which presents some historical data on clients such as The number of days the borrower has had a credit line, the borrower's number of inquiries by creditors in the last 6 months and other information.

Before beginning the construction of the deep learning model, we first carry out the EAD and feature engineering operations. In the next step we look at the performance of some machine learning algorithms to consider them as a basis for reflection on our deep learning model to be built. Before attacking our objective, we will try to reduce the features by referring to feature_importances_ of RandomForestClassifier.

To build our deep learning model, we will use two overfitting techniques **Dropout Regularization and l2 Regularization**. For more efficiency, the operation which consists of to lower the learning rate as the training progresses is ensured by **the inverseTimeDecay** function. And to finalize our model we will use a **Callback** function with a target.

the Python libraries that will be useful to us can be found at the end of this document with this link, and after loading the dataset :

```
import pandas as pd
df_init = pd.read_csv('/content/loan_data.csv')
#
df=df_init.copy(deep=True)
# Top 5 records
df.head()
```

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delinq.2yrs	pub.rec	not.fully.paid
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	52.1	0	0	0	0
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	76.7	0	0	0	0
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710.000000	3511	25.6	1	0	0	0
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2699.958333	33667	73.2	1	0	0	0
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	4066.000000	4740	39.5	0	1	0	0

We give some necessary information to better know our dataset:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   credit.policy      9578 non-null   int64  
 1   purpose            9578 non-null   object 
 2   int.rate           9578 non-null   float64
 3   installment        9578 non-null   float64
 4   log.annual.inc     9578 non-null   float64
 5   dti                9578 non-null   float64
 6   fico               9578 non-null   int64  
 7   days.with.cr.line  9578 non-null   float64
 8   revol.bal          9578 non-null   int64  
 9   revol.util         9578 non-null   float64
 10  inq.last.6mths     9578 non-null   int64  
 11  delinq.2yrs        9578 non-null   int64  
 12  pub.rec            9578 non-null   int64  
 13  not.fully.paid     9578 non-null   int64  
dtypes: float64(6), int64(7), object(1)
memory usage: 1.0+ MB
```

We have, therefore, a dataset with 9578 entries and without null value and 14 columns. Now let's analyze the type of these columns :

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

```
int64      7
float64    6
object     1
dtype: int64
```

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

	purpose
0	debt_consolidation
1	credit_card
2	debt_consolidation
3	debt_consolidation
4	credit_card

A single column of type 'object':

```
df['purpose'].value_counts()
```

```
debt_consolidation    3957
all_other             2331
credit_card           1262
home_improvement       629
small_business         619
major_purchase         437
educational            343
Name: purpose, dtype: int64
```

For the following we will use the function **get_value_counts()** which gives for a given type the name of values that appear in the columns of this type.

```
# type= object
lst_t=get_value_counts(df,'object')
lst_t
```

```
[('purpose', 7)]
```

We will then have to do the transformation categorical values into numerical values of this column. Before we analyze the other types of columns :

```
# int64
df.select_dtypes('int64').head()
```

	credit.policy	fico	revol.bal	inq.last.6mths	delinq.2yrs	pub.rec	not.fully.paid
0	1	737	28854	0	0	0	0
1	1	707	33623	0	0	0	0
2	1	682	3511	1	0	0	0
3	1	712	33667	1	0	0	0
4	1	667	4740	0	1	0	0

```
# type= int64
lst_t=get_value_counts(df,'int64')
lst_t
```

```
[('credit.policy', 2),
 ('fico', 44),
 ('revol.bal', 7869),
 ('inq.last.6mths', 28),
 ('delinq.2yrs', 11),
 ('pub.rec', 6),
 ('not.fully.paid', 2)]
```

```
# type= float64
lst_t=get_value_counts(df,'float64')
lst_t
```

```
[('int.rate', 249),
 ('installment', 4788),
 ('log.annual.inc', 1987),
 ('dti', 2529),
 ('days.with.cr.line', 2687),
 ('revol.util', 1035)]
```

In order to start studying the models let's apply the **get_dummies()** function of pandas and analyze the result :

```
df1=pd.get_dummies(df)
df1.head()
```

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths
0	1	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	52.1	0
1	1	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	76.7	0
2	1	0.1357	366.86	10.373491	11.63	682	4710.000000	3511	25.6	1
3	1	0.1008	162.34	11.350407	8.10	712	2699.958333	33667	73.2	1
4	1	0.1426	102.92	11.299732	14.97	667	4066.000000	4740	39.5	0

```
df1.dtypes.value_counts()
```

```
int64      7
float64    6
object     1
dtype: int64
```

```
# reminder
df.dtypes.value_counts()
```

```
int64      7
float64    6
object     1
dtype: int64
```

```
# uint8
df1.select_dtypes('uint8').columns
```

```
Index(['purpose_all_other', 'purpose_credit_card',
       'purpose_debt_consolidation', 'purpose_educational',
       'purpose_home_improvement', 'purpose_major_purchase',
       'purpose_small_business'],
      dtype='object')
```

We had the replacement of the column purpose by these columns whose name is prefixed by purpose.

```
# type= uint8
lst_t=get_value_counts(df1,'uint8')
lst_t
```

```
[('purpose_all_other', 2),
 ('purpose_credit_card', 2),
 ('purpose_debt_consolidation', 2),
 ('purpose_educational', 2),
 ('purpose_home_improvement', 2),
 ('purpose_major_purchase', 2),
 ('purpose_small_business', 2)]
```

Only values 0 and 1 for these columns :

```
# type= uint8
lst_t=df1.select_dtypes('uint8').values
ll=[]
for i in range(len(lst_t)):
  ll=ll+list(lst_t[i,:])
set(ll)
```

```
{0, 1}
```

We define the X and the Y (Target) to then, define also, the Train and Validation data:

```
# def of X, Y
Y=df1[df1.columns[0]]
X=df1[df1.columns[1:]]
X.shape, Y.shape,len(df1),len(df1.columns)
```

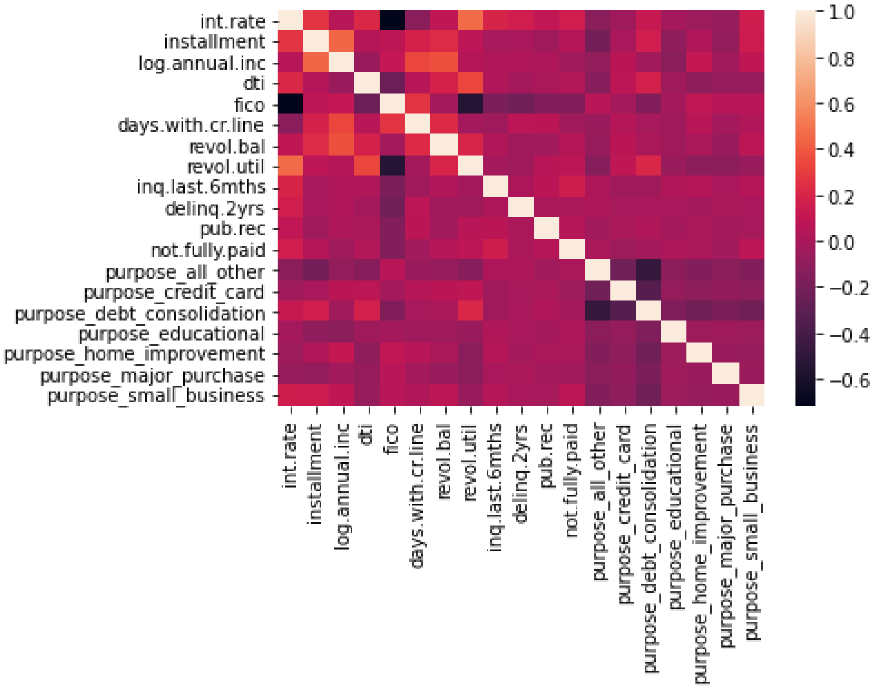
```
((9578, 19), (9578,), 9578, 20)
```

We check features dependency :

```
# Significant dependency
corr_matrix_X = X.corr()
print('corr_matrix_X.shape')
print(corr_matrix_X.shape)
#
coef_dep=0.85
#
if len(verify_dependency(corr_matrix_X,coef_dep))==0:
  print('\nNo dependency:', verify_dependency(corr_matrix_X,coef_dep))
else:
  print('\nDependency:....', verify_dependency(corr_matrix_X,coef_dep))
#
print('\n')
import seaborn as sns
```

```
sns.heatmap(corr_matrix_X)
```

```
corr_matrix_X.shape
(19, 19)

No dependency: []
```



At this level we split the data with 25% the validation data:

```
# Split() ---> define X_train,X_test,y_train,y_test
from sklearn.model_selection import train_test_split
#
in_test_size=25/100
X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=in_test_size,random_state=0)
#
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

```
((7183, 19), (2395, 19), (7183,), (2395,))
```

Before the build of our deep learning model, we look at the performance of some classical machine learning models by referring to function [test_classifierModels_list()]():

```
# list classifiers
lst_algo=['AdaBoost','SVM','KNN','DecisionTreeClassifier','RandomForestClassifier']
#
dico_name_model=test_classifierModels_list(lst_algo,X_train,X_test,y_train,y_test)
#dico_name_model
```

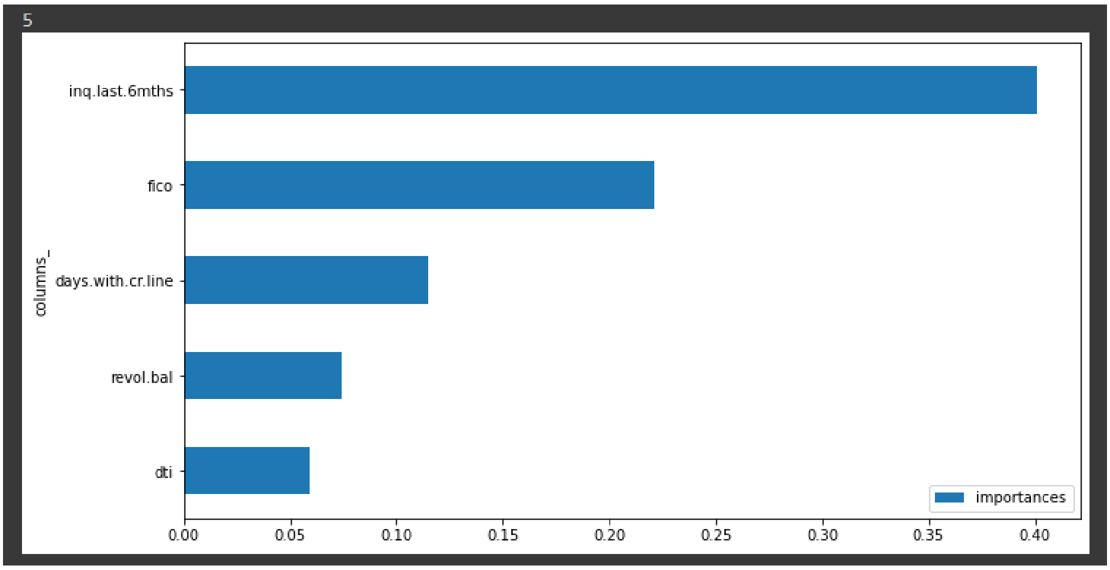
```
============
RandomForestClassifier
============
[[ 399   80]
 [   2 1914]]
              precision    recall  f1-score   support

           0     0.9950    0.8330    0.9068       479
           1     0.9599    0.9990    0.9790      1916

    accuracy                         0.9658      2395
   macro avg     0.9774    0.9160    0.9429      2395
weighted avg     0.9669    0.9658    0.9646      2395

============
AdaBoost
============
[[ 364  115]
 [  17 1899]]
              precision    recall  f1-score   support

           0     0.9554    0.7599    0.8465       479
           1     0.9429    0.9911    0.9664      1916

    accuracy                         0.9449      2395
   macro avg     0.9491    0.8755    0.9065      2395
weighted avg     0.9454    0.9449    0.9424      2395
```

```
============
SVM
============
[[ 324  155]
 [   8 1908]]
              precision    recall  f1-score   support

           0     0.9759    0.6764    0.7990       479
           1     0.9249    0.9958    0.9590      1916

    accuracy                         0.9319      2395
   macro avg     0.9504    0.8361    0.8790      2395
weighted avg     0.9351    0.9319    0.9270      2395

============
KNN
============
[[ 346  133]
 [  13 1903]]
              precision    recall  f1-score   support

           0     0.9638    0.7223    0.8258       479
           1     0.9347    0.9932    0.9631      1916

    accuracy                         0.9390      2395
   macro avg     0.9492    0.8578    0.8944      2395
weighted avg     0.9405    0.9390    0.9356      2395
```

```
============
DecisionTreeClassifier
============
[[ 407   72]
 [  39 1877]]
              precision    recall  f1-score   support

           0     0.9126    0.8497    0.8800       479
           1     0.9631    0.9796    0.9713      1916

    accuracy                         0.9537      2395
   macro avg     0.9378    0.9147    0.9256      2395
weighted avg     0.9530    0.9537    0.9530      2395
```

In order to reduce the analysis features we will use the classifier RandomForestClassifier() and its feature importances, we refer to the **show_importances_plot()** function:

```
model_rfc = RandomForestClassifier(random_state=10)
model_rfc_fit= model_rfc.fit(X_train,y_train)
#
#-----------------------------------------
#--- feature_importances ---> for RandomForestClassifier
#-----------------------------------------
#
feature_importances_rfc=show_importances_plot(model_rfc_fit,i_threshold=0.05) #0.01
len(feature_importances_rfc)
```



We take the features that correspond to values of importance greater than the threshold 0.5 :

```
#columns list
list(feature_importances_rfc.index
```

```
['dti', 'revol.bal', 'days.with.cr.line', 'fico',
'inq.last.6mths']
```

```
list_rest_cols=list(feature_importances_rfc.index)
#
X_train1=X_train[list_rest_cols]
X_test1=X_test[list_rest_cols]
y_train1=y_train
y_test1=y_test
#
model_rfc = RandomForestClassifier(random_state=10)
model_rfc_fit1= model_rfc.fit(X_train1,y_train1)
model_rfc_fit1.score(X_test1,y_test1)
```

```
0.9899791231732776
```

By restricting ourselves to these **five features,** the performance does not decrease and therefore, **for the rest we remain on this consideration**:

```
X_train1.shape,y_train1.shape,X_test1.shape,y_test1.shape
```

```
((7183, 5), (7183,), (2395, 5), (2395,))
```

We check the keras backend used :

```
# backend keras = ? tensorflow
import keras
keras.backend.backend()
```

```
tensorflow
```

We standardize the data :

```
# X_train1,X_test1,y_train1,y_test1
X_train2=X_train1.copy(deep=True)
X_test2=X_test1.copy(deep=True)
y_train2=y_train1.copy(deep=True)
y_test2=y_test1.copy(deep=True)
#
scaler2=StandardScaler()
scaler2.fit(X_train2)
#
X_train2 = scaler2.transform(X_train2)
X_test2 = scaler2.transform(X_test2)
```

The definition of the callbacks class which will be used to stop the training once the performance objective has been reached, and to display the intermediate performance after each 1000 epochs :

```
# Callback
class myCallback(tf.keras.callbacks.Callback):
 max_val_Acc=0

 def __init__(self,i_threshold):
  self.threshold_cb=i_threshold

 def on_epoch_end(self, epoch, logs={}):
    if (logs.get('val_accuracy') > self.threshold_cb):
        print("\nReached ", self.threshold_cb*100 ,
            "% val_accuracy so cancelling training!")
        print("\nepoch: ", epoch)
        print('Acc=', logs.get('accuracy'))
        print('val_Acc=', logs.get('val_accuracy'))
        print('loss=', logs.get('loss'))
        print('\n')
        self.model.stop_training = True
    if (epoch%1000==0):
      print("\nepoch: ", epoch)
```

```
    print('Acc=', logs.get('accuracy'))
    print('val_Acc=', logs.get('val_accuracy'))
    if logs.get('val_accuracy') > self.max_val_Acc:
      self.max_val_Acc= logs.get('val_accuracy')
    print('max_val_Acc=', self.max_val_Acc)
    print('loss=', logs.get('loss'))
```

for more efficiency we have defined the **InverseTimeDecay** object, and the function **get_optimizer()** to set the Adam optimizer parameters :

```
# Building model preparation
#N_VALIDATION = int(1e3)
N_TRAIN = int(1e4)
BUFFER_SIZE = int(1e4)
BATCH_SIZE = 1024
STEPS_PER_EPOCH = N_TRAIN//BATCH_SIZE

# InverseTimeDecay
lr_schedule = tf.keras.optimizers.schedules.InverseTimeDecay(
  0.001,
  decay_steps=STEPS_PER_EPOCH*300, # 1000
  decay_rate=1,
  staircase=False)

# optimizer
def get_optimizer(i_lr_schedule):
  l_adam= tf.keras.optimizers.Adam (
      learning_rate=i_lr_schedule,
      beta_1=0.9,
      name='adam'
  )
  return l_adam
```

Now, we have all the elements to launch the training with the definition of the model which is given by get_model():

Model with 3 layers :

- Dense(1024, input_dim=i_input_dim,activation='relu', kernel_regularizer=l2(i_regul_l2)))
- Dropout(0.7)
- Dense(1,activation='sigmoid'))

```
# Building model
#
target_score=0.98
callbacks = myCallback(target_score)

model_tf = get_model(X_train2.shape[1],i_def_mode=1)

model_tf.compile(loss='binary_crossentropy', optimizer=get_optimizer(lr_schedule),
          metrics=['accuracy'])

import time
start=time.time()

history=model_tf.fit(X_train2,y_train2,epochs=50000,batch_size=BATCH_SIZE,
              validation_data=(X_test2, y_test2) ,callbacks=[callbacks],verbose=0)

end=time.time()
print('running time: ',end-start)

score=model_tf.evaluate(X_test2,y_test2,batch_size=BATCH_SIZE)

print('\nscore: ',score,'\n')
```

```
epoch:  4000
Acc= 0.9700682163238525
val_Acc= 0.9774530529975891
max_val_Acc= 0.9774530529975891
loss= 0.10118568688631058

epoch:  5000
Acc= 0.9722957015037537
val_Acc= 0.9774530529975891
max_val_Acc= 0.9774530529975891
loss= 0.09867680817842484

Reached  98.0 % val_accuracy so cancelling training!

epoch:  5939
Acc= 0.973827064037323
val_Acc= 0.9803757667541504
loss= 0.09569501876831055


running time:  1048.621102809906
3/3 [==============================] - 0s 7ms/step - loss: 0.1025 - accuracy: 0.9804

score:  [0.10246644914150238, 0.9803757667541504]
```

The goal for accuracy to exceed the 98% threshold was reached after 5939 epochs. We give below the confusion matrix and other classification metrics such as precision and recall. We clearly see a good result with balanced metrics.

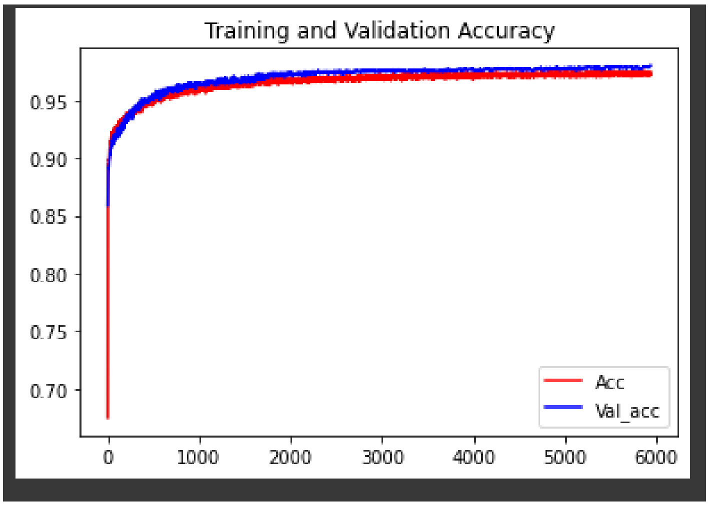
```
print('score: ',score,'\n')
model= fit_and_evaluate_model(model_tf,X_train2,X_test2,y_train2,y_test2,i_use_fit
=False,cas_tensorflow=True)
```

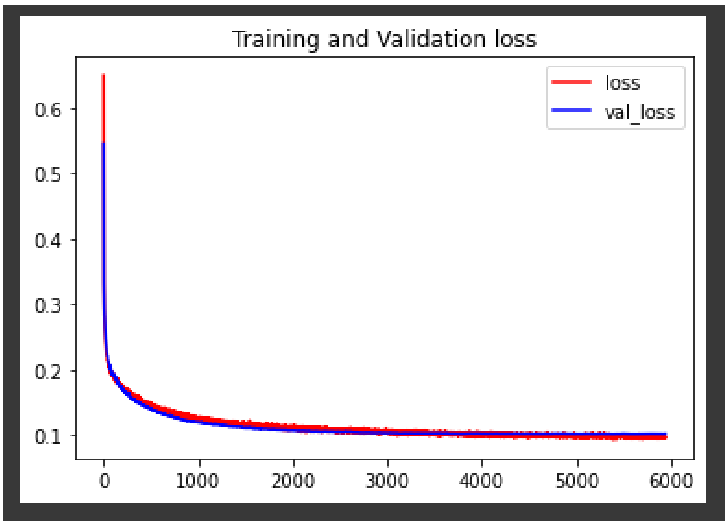
```
score:  [0.10246644914150238, 0.9803757667541504]

[[ 447   32]
 [  15 1901]]
              precision    recall  f1-score   support

           0     0.9675    0.9332    0.9501       479
           1     0.9834    0.9922    0.9878      1916

    accuracy                         0.9804      2395
   macro avg     0.9755    0.9627    0.9689      2395
weighted avg     0.9803    0.9804    0.9802      2395
```

We finish our presentation by displaying the evolution of the accuracy and the loss by the advancement of the epochs :

```
#-----------------------------------------------
# Plot training and validation accuracy per epoch
#-----------------------------------------------
acc=history.history['accuracy']
val_acc=history.history['val_accuracy']

epochs=range(len(acc)) # Get number of epochs

plt.plot(epochs, acc, 'r')
plt.plot(epochs, val_acc, 'b')
#
plt.title('Training and Validation Accuracy')
plt.legend(['Acc','Val_acc'])
plt.show()
print("")

#-----------------------------------------------
# Plot training and validation loss per epoch
#-----------------------------------------------
loss=history.history['loss']
val_loss=history.history['val_loss']
epochs=range(len(loss)) # Get number of epochs
#
plt.plot(epochs, loss, 'r')
```

```
plt.plot(epochs, val_loss, 'b')
#
plt.title('Training and Validation loss')
plt.legend(['loss','val_loss'])
plt.show()
print("")
```





FUNCTIONS

Librairies

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, Dropout
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
from keras.regularizers import l2
from sklearn.model_selection import train_test_split

# list classifiers
lst_algo=['AdaBoost','SVM','KNN','DecisionTreeClassifier','RandomForest
Classifier']
```

test_classifierModels_list() function

```
#######################################
### 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
```

fit_and_evaluate_model () function

```
###########################
### Evaluation function: fit_and_evaluate_model()
##########################
# Used to Evaluate models
def fit_and_evaluate_model(i_model,i_X_train,i_X_test,i_y_train,i_y_tes
t, i_use_fit=True,cas_tensorflow=False):
  #
  l_model=i_model
  if not cas_tensorflow:
    if i_use_fit:
      l_model.fit(i_X_train, i_y_train)
  l_ypred = l_model.predict(i_X_test)
  #
```

```
  if cas_tensorflow:
    l_ypred_t=np.array([0 for i in range(len(l_ypred))])
    for i in range(len(l_ypred)):
       if l_ypred[i] > 0.5:
         l_ypred_t[i]=1
    l_ypred=l_ypred_t
  #
  print(confusion_matrix(i_y_test, l_ypred))
  print(classification_report(i_y_test, l_ypred, digits=4))

  return l_model
```

get_value_counts() function

```
# 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
```

show_importances_plot() function

```
#---------------------------------
#--- 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_na
mes_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 feature_importances_df_t
```

verify_dependency() function

```
# return list of dep columns couple
def verify_dependency(i_corr_matrix,i_threshold):
  # looking for features corresponding at the threshold 0.35
  cols=i_corr_matrix.columns
  l_tupe_dep=[]  # list tuple of columns to verify dep condition
  l_cmpt=0
  for col1 in cols:
    for col2 in cols:
      l_val=i_corr_matrix.loc[col2,col1]
      #if l_cmpt < 10:
        #print (abs(l_val))
        #l_cmpt=l_cmpt+1
      if abs(l_val) > i_threshold:
        if col1!=col2:
          if (col1,col2) not in l_tupe_dep:
            l_tupe_dep.append((col1,col2))

  return l_tupe_dep
```

get_model() function

```
#----------------------------------
#--- function get_model()
# give model with appropriate input_dim as parameter
#----------------------------------
#
def get_model(i_input_dim,i_regul_l2=0.0001,i_def_mode=1):
  l_model=Sequential()
  if i_def_mode==1:
    l_model.add(Dense(1024, input_dim=i_input_dim,activation='relu',
     kernel_regularizer=l2(i_regul_l2)))
           l_model.add(Dropout(0.7))
  #
```

```
if i_def_mode==2:
   l_model.add(Dense(1278, input_dim=i_input_dim,activation='relu',
     kernel_regularizer=l2(i_regul_l2)))
            l_model.add(Dropout(0.5))
   l_model.add(Dense(512, activation='elu'),
                          kernel_regularizer=l2(i_regul_l2)),
   l_model.add(Dropout(0.5))
l_model.add(Dense(1,activation='sigmoid'))
#
```