Rachid LAMJOUN
AI & ML Master Program

E-commerce

(AI Capstone Project: Project1 E-commerce)

Writeup

We have the subject of predicting Sentiment Amazon customers by website. The data to be studied, represents thousands of consumer reviews for Amazon branded products such as Kindle, Fire TV Stick etc.

We have the task to predict three sentiment levels Positive, Negative and Neutral. What is interesting is that we have training data from our models separated from validation data. Which is also separated from sentiments corresponding to real consumer opinions. The validation is therefore done with data that has never been seen and the predictions made on this validation data have never seen the real feedback sentiment from consumers.

In our search for the best model, we will follow this trajectory! by answering or successively carrying out the points below:

I- Class Imbalance Problem:

1. P1. Seeing what a positive, negative, and neutral review looks like
2. P2. Checking the class count for each class. Identification of imbalance class
3. P3. Conversion of reviews in Tf-Idf score
4. P4. Running multinomial Naive Bayes classifier. demonstration of class imbalance problem with unique positive prediction!!

II- Tackling Class Imbalance Problem:

5. P5. Tackle the class imbalance problem by using the technique of Oversampling or undersampling
6. P6. Metrics for evaluation:

   As we have class imbalance problem, we use those metrics for evaluating model performance: precision, recall, F1-score, and AUC-ROC curve. F1-Score metric will be used as a principal evaluation criteria for this project the first task is to check the consistency of the datasets in our possession. the datasets **train_data.csv** and **test_data_hidden.csv** have the same columns while **test_data.csv** don't have the target column 'sentiment'. it was necessary to verify that the difference between the test_data_hidden.csv and test_data.csv datasets in data and columns is only this 'sentiment' column which is additionally in test_data_hidden.csv. Also check that the sentiment columns of train_data.csv and test_data_hidden.csv contain the same values. For details see the EDA Analysis paragraph in the detail section below.

By answering the questions of the first paragraph "Class Imbalance Problem" we highlighted a class imbalance that we solved by the **oversampling** which was possible by **RandomOverSampler** of **imblearn.over_sampling**. And without optimization, we had the following results for the different models tested in our subject:
For this table **oversampling** was used except when the specific option was used for Classifier.

	precision	recall	f1-score	accuracy
BaggingClassifier() base_estimator=RandomForestClassifier()	0.99	0.53	0.63	0.96
BaggingClassifier() base_estimator= XGBClassifier ()	0.54	0.72	0.60	0.89
RandomForestClassifier class_weight='balanced'	0.98	0.44	0.51	0.95
RandomForestClassifier	0.98	0.52	0.61	0.95
XGBClassifier (scale_pos_weight=40)	0.80	0.42	0.47	0.94
XGBClassifier	0.77	0.63	0.69	0.90
SVC	0.99	0.54	0.64	0.96
MultinomialNB	0.50	0.63	0.54	0.88
Dense Layer	0.75	0.59	0.65	0.95
GRU Layer	0.69	0.60	0.63	0.94
LSTM Layer	0.36	0.42	0.38	0.90
Bidirectional LSTM Layers	0.66	0.64	0.64	0.94

We note that the neural network models are doing well except for the one made up of simple LTSM layers. We note that the two models XGBoost and Bidirectional LSTM give balanced results with respect to all types of score.

For optimization, we focused on two models XGBoost and Bidirectional LSTM and we had the following results:

	precision	recall	f1-score	accuracy
XGBClassifier optimisé	**0.78**	**0.65**	**0.70**	**0.96**
Bidirectional LSTM Layers	0.73	0.60	0.64	0.95

Even other optimizations are still possible we note that XGGBoost gives the best results and relatively balanced.

Concerning the subject of Topic Modeling, we used for the LDA approach, the **ldamodel** from **gensim.models** and which generated for 8 topics and 12-words topn the following array:

	Topic # 01	Topic # 02	Topic # 03	Topic # 04	Topic # 05	Topic # 06	Topic # 07	Topic # 08
0	love	tablet	echo	great	love	love	kindl	use
1	great	great	tablet	amazon	one	bought	tablet	great
2	use	good	love	music	use	tablet	love	love
3	echo	use	great	use	purchas	like	great	easi
4	alexa	easi	use	play	bought	use	one	product
5	amazon	love	get	video	got	set	fire	read
6	show	price	devic	alexa	tablet	nice	old	bought
7	like	kid	amazon	product	new	play	bought	alexa
8	tablet	screen	need	app	realli	good	use	light
9	one	would	buy	love	enjoy	time	read	work
10	kindl	recommend	plus	echo	gift	year	year	book
11	thing	read	want	tablet	kindl	product	purchas	play

And for the NMF approach we used the **NMF** model of **sklearn.decomposition** and we had a generation as before for 8 topics and 12 words the following table:

	Topic # 01	Topic # 02	Topic # 03	Topic # 04	Topic # 05	Topic # 06	Topic # 07	Topic # 08
0	great	love	easi	tablet	echo	good	old	kindl
1	work	bought	use	kid	alexa	product	year	read
2	price	gift	set	price	show	recommend	grandson	book
3	product	daughter	product	need	music	would	bought	fire
4	kid	son	fun	app	home	price	perfect	game
5	gift	absolut	setup	perfect	like	friend	purchas	like
6	sound	christma	super	game	one	buy	one	replac
7	valu	got	learn	nice	plus	qualiti	christma	play
8	well	granddaught	navig	amazon	amazon	high	enjoy	size
9	camera	kid	problem	daughter	light	excel	son	one
10	addit	wife	item	littl	screen	definit	yr	better
11	deal	grandson	simpl	play	smart	time	happi	purchas

DETAILS

I- Class Imbalance Problem

EDA Analysis

```
import pandas as pd
# Loading train_data, test_data and test_data_hidden
df_init = pd.read_csv('/content/train_data.csv')
df_test_init = pd.read_csv('/content/test_data.csv')
df_htest_init = pd.read_csv('/content/test_data_hidden.csv')
#
df=df_init.copy(deep=True)
df_test=df_test_init.copy(deep=True)
df_htest=df_htest_init.copy(deep=True)
#

# train_data: Top 5 records
df.head()


# train_data: info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4000 entries, 0 to 3999
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   name               4000 non-null   object
 1   brand              4000 non-null   object
 2   categories         4000 non-null   object
 3   primaryCategories  4000 non-null   object
 4   reviews.date       4000 non-null   object
 5   reviews.text       4000 non-null   object
 6   reviews.title      3990 non-null   object
 7   sentiment          4000 non-null   object
dtypes: object(8)
memory usage: 250.1+ KB
```

```
# test_data: info
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 7 columns):
```

```
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   name               1000 non-null   object
 1   brand              1000 non-null   object
 2   categories         1000 non-null   object
 3   primaryCategories  1000 non-null   object
 4   reviews.date       1000 non-null   object
 5   reviews.text       1000 non-null   object
 6   reviews.title      997 non-null    object
dtypes: object(7)
memory usage: 54.8+ KB
```

```
# test_data_hidden: info
df_htest.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   name               1000 non-null   object
 1   brand              1000 non-null   object
 2   categories         1000 non-null   object
 3   primaryCategories  1000 non-null   object
 4   reviews.date       1000 non-null   object
 5   reviews.text       1000 non-null   object
 6   reviews.title      997 non-null    object
 7   sentiment          1000 non-null   object
dtypes: object(8)
memory usage: 62.6+ KB
```

```
# shape
df.shape,df_test.shape,df_htest.shape
```

```
((4000, 8), (1000, 7), (1000, 8))
```

The same data for test_data and test_data_hidden, only that the sentiment column does not exist for test_data

```
#---------------------------------
# same data test and htest ?
#----------------------------------
# df_htest.loc[:, df_htest.columns!='sentiment']
df_htest_no_target=df_htest.loc[:, df_htest.columns[:-1]]
#
#df_htest_no_target.info()
df_htest_no_target[df_htest_no_target==df_test].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   name               1000 non-null   object
```

```
 1   brand             1000 non-null   object
 2   categories        1000 non-null   object
 3   primaryCategories 1000 non-null   object
 4   reviews.date      1000 non-null   object
 5   reviews.text      1000 non-null   object
 6   reviews.title     997 non-null    object
dtypes: object(7)
memory usage: 54.8+ KB
```

```
len(df_htest_no_target[df_htest_no_target==df_test]),len(df_htest_no_target
)
```

```
(1000, 1000)
```

The same value for columns sentiment of train_data and test_data_hidden

```
df.sentiment.value_counts()
```

```
Positive    3749
Neutral      158
Negative      93
Name: sentiment, dtype: int64
```

```
df_htest.sentiment.value_counts()
```

```
Positive    937
Neutral      39
Negative     24
Name: sentiment, dtype: int64
```

Dropping missing values

```
# shape
df.shape,df_test.shape,df_htest.shape
```

```
((4000, 8), (1000, 7), (1000, 8))
```

```
# Dropping missing values
df.dropna(inplace=True)
df_test.dropna(inplace=True)
df_htest.dropna(inplace=True)
#
df=df.reset_index()
df_test=df_test.reset_index()
df_htest=df_htest.reset_index()
#
# shape
df.shape,df_test.shape,df_htest.shape
```

```
((3990, 9), (997, 8), (997, 9))
```

P1. Seeing what a positive, negative, and neutral review looks like

```
# these are previews in english
import random
#
for i in range(5):
  n = random.randint(0,len(df))
  print(df['reviews.title'].loc[n])
  print(len(df['reviews.text'].loc[n]),'     ',df['reviews.text'].loc[n],'\
n')
```

```
Good tablet for basic purposes
99       I got this tablet for Skype calls and browsing. It's good if you d
on't have an ample scope of tasks

Lots of problems. Want my old one back
208       1st kindle screen failed & had to reboot often. Customer Service
couldn't fix. Replaced. 2nd one screen reboots often. Customer service offe
red replace. Just want one that works. Can't get upgrade replacement

Awesome tablet
58       So far it does what I want. Very happy with the price too.

Greater starter for my 2 yr. old grandson
64       Easy and simple for my grandson to use. Has no problem using it.

ECHO PLUS GET ONE
155       The whole family uses the ECHO we love the ability to use just sa
y Alexa and tell here what we want. The youngest granddaughter loves the in
sult generator!
```

```
df.index
```

```
RangeIndex(start=0, stop=3990, step=1)
```

```
# df_t DataFrame temp
df_t=pd.DataFrame(index=range(0,len(df)))
df_t['len_preview_title']=0
df_t['len_preview_title'].loc[340]
```

```
0
```

```
# df_t DataFrame temp
df_t=pd.DataFrame(index=range(0,len(df)))
df_t['len_preview_title']=0
df_t['len_preview_text']=0
#
for i in range(0,len(df)):
  df_t['len_preview_title'].loc[i]=len(df['reviews.title'].loc[i])
  df_t['len_preview_text'].loc[i]=len(df['reviews.text'].loc[i])

print('df_t[\'len_preview_title\'].max()=',df_t['len_preview_title'].max())
print('df_t[\'len_preview_text\'].max()=',df_t['len_preview_text'].max())
```

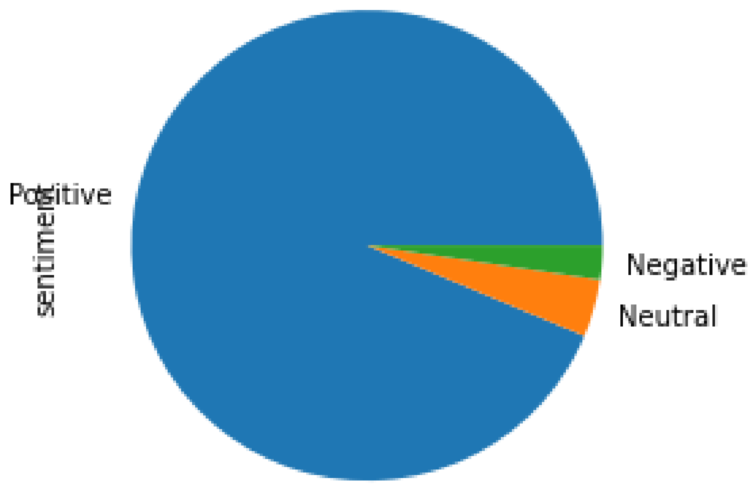
```
df_t['len_preview_title'].max()= 71
df_t['len_preview_text'].max()= 8351
```

P2. Checking the class count for each class. Identification of imbalance class

Unbalanced data: Unbalanced with the target 'sentiment'

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5874ef8a50>
```




```
for val in ['Positive','Negative','Neutral']:
  df_t=df[df['sentiment']==val]
  print(val)
  print('     >>',round(len(df_t)/len(df),2),'%')
```

```
Positive
     >> 0.94 %
Negative
     >> 0.02 %
Neutral
     >> 0.04 %
```

P3. Conversion of reviews in Tf-Idf score

```
# using function init_data_treatment()
sentences,labels,X,X_test_for_htest,df_htest=init_data_treatment(i_txt_proc
ess=2,i_epoch=5,i_ROS_op=False)
```

```
X.head()
```

```
                                  reviews_title_text
0  powerful tablet purchased on black fridaypros ...
1  amazon echo plus awesome i purchased two amazo...
2  average just an average alexa option does show...
3  greatttttttt very good product exactly what i w...
4  very durable this is the 3rd one ive purchased...
```

```
X_test_for_htest.head()
```

```
                                  reviews_title_text
0  very handy device amazon kindle fire has a lot...
1  another winner from amazon the echo show is a ...
2  simple to use and reliable so far great value ...
3  love it i use mine for email facebook games an...
4  fantastic this is a fantastic item the person ...
```

```
df_htest.head()
```

```
                                                name   brand  \
0  Fire Tablet, 7 Display, Wi-Fi, 16 GB - Include...  Amazon
1  Amazon Echo Show Alexa-enabled Bluetooth Speak...  Amazon
2  All-New Fire HD 8 Tablet, 8" HD Display, Wi-Fi...  Amazon
3  Brand New Amazon Kindle Fire 16gb 7" Ips Displ...  Amazon
4  Amazon Echo Show Alexa-enabled Bluetooth Speak...  Amazon

                                          categories     primaryCategories
\
0  Fire Tablets,Computers/Tablets & Networking,Ta...           Electronics
1  Computers,Amazon Echo,Virtual Assistant Speake...  Electronics,Hardware
2  Electronics,iPad & Tablets,All Tablets,Fire Ta...           Electronics
3  Computers/Tablets & Networking,Tablets & eBook...           Electronics
4  Computers,Amazon Echo,Virtual Assistant Speake...  Electronics,Hardware

               reviews.date  \
0  2016-05-23T00:00:00.000Z
1  2018-01-02T00:00:00.000Z
2  2017-01-02T00:00:00.000Z
3  2017-03-25T00:00:00.000Z
4  2017-11-15T00:00:00.000Z

                                        reviews_text  \
0  Amazon kindle fire has a lot of free app and c...
1  The Echo Show is a great addition to the Amazo...
2  Great value from Best Buy. Bought at Christmas...
3  I use mine for email, Facebook ,games and to g...
4  This is a fantastic item & the person I bought...

                     reviews_title  sentiment
0                very handy device          2
1           Another winner from Amazon          2
2  simple to use and reliable so far          2
3                         Love it!!!          2
4                         Fantastic!          2
```

```
# processed preview
sentences[:5]
```

```
['powerful tablet purchased on black fridaypros great price even off saleve
ry powerful and fast with quad core processors amazing soundwell builtcons
amazon ads amazon need this to subsidize the tablet and will remove the add
s if you pay them 15inability to access other apps except the ones from ama
zon there is a way which i was able to accomplish to add the google play st
```

```
orenet this is a great tablet for the money',
 'amazon echo plus awesome i purchased two amazon in echo plus and two dots
plus four fire sticks and the hub philips hue for lamp for the family at ch
ristmas 2017 i,äôm so happy with these purchases and learning so much with
alexa you can start your daily routine with alexa and program it to whateve
r you would like to include news weather music horoscope also you can start
your day off with a compliment and i think is very important alexa gave me
the best chili recipe i mean the best it,äôs called chili i i want my husba
nd to use alexa to stay organized for business dates and reminders this is
the way to go',
 'average just an average alexa option does show a few things on screen but
still limited',
 'greattttttt very good product exactly what i wanted and a very good price
',
 'very durable this is the 3rd one ive purchased ive bought one for all of
my nieces no other case compares to this one it has held protected the tabl
et so many times from them dropping it']
```

```
# sentiment
labels[:5]
```

```
[2, 2, 1, 2, 2]
```

```
# checking
len(X_test_for_htest),len(df_htest)
```

```
(997, 997)
```

```
# checking
len(X),len(sentences),len(labels)
```

```
(3990, 3990, 3990)
```

```
# Data preparation for fit()
y=pd.Series(labels)
if isinstance(X, pd.core.frame.DataFrame):
  X=X['reviews_title_text'] # X must be Series for train after
Xtest_h=X_test_for_htest['reviews_title_text']
ytest_h=df_htest['sentiment']
#
Xtest_h.shape,ytest_h.shape,y.shape
#
# Split() X.....
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, r
andom_state=101)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((2992,), (998,), (2992,), (998,))
```

```
# Tf-idf operation
from sklearn.feature_extraction.text import TfidfVectorizer
tf_idf = TfidfVectorizer()
#
# conversion of reviews in Tf-Idf score
X_train_tfidf = tf_idf.fit_transform(X_train)
```

```
X_test_tfidf=tf_idf.transform(X_test)
#print('X_test_tfidf.shape=',X_test_tfidf.shape)
```

P4. Running multinomial Naive Bayes classifier. demonstration of class imbalance problem with unique positive prediction!!

```
from sklearn.naive_bayes import MultinomialNB
model_mnb = MultinomialNB()
#
model_mnb.fit(X_train_tfidf,y_train)
model_mnb.score(X_train_tfidf,y_train),model_mnb.score(X_test_tfidf,y_test)
```

```
(0.9364973262032086, 0.938877755511022)
```

```
# test for df_test et df_htest datasets
Xtest_h=X_test_for_htest['reviews_title_text']
model_mnb.score(tf_idf.transform(Xtest_h),df_htest['sentiment'])
```

```
0.9368104312938816
```

```
# all prediction are Positive!
predict_test_h=model_mnb.predict(tf_idf.transform(Xtest_h))
predict_test_h_unique=set(predict_test_h)
predict_test_h_unique
```

```
{2}
```

II- Tackling Class Imbalance Problem

P5. Tackle the class imbalance problem by using the technique of Oversampling or undersampling

```
# using function init_data_treatment()
# imblearn.over_sampling.RandomOverSampler to handle imbalanced data
# ===> i_ROS_op=True

sentences,labels,X,X_test_for_htest,df_htest=init_data_treatment(i_txt_proc
ess=2,i_epoch=5,i_ROS_op=True)
```

```
# checking
len(X_test_for_htest),len(df_htest)
```

```
(997, 997)
```

```
# checking
len(X),len(sentences),len(labels)
```

```
(11217, 11217, 11217)
```

```
# Data preparation for fit()
y=pd.Series(labels)
if isinstance(X, pd.core.frame.DataFrame):
  X=X['reviews_title_text'] # X must be Series for train after
```

```
Xtest_h=X_test_for_htest['reviews_title_text']
ytest_h=df_htest['sentiment']
#
Xtest_h.shape,ytest_h.shape,y.shape
#
# Split() X.....
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, r
andom_state=101)
X_train.shape, X_test.shape, y_train.shape, y_test

# Tf-idf operation
from sklearn.feature_extraction.text import TfidfVectorizer
tf_idf_ROS = TfidfVectorizer()
#
# conversion of reviews in Tf-Idf score
X_train_tfidf = tf_idf_ROS.fit_transform(X_train)
X_test_tfidf=tf_idf_ROS.transform(X_test)
#print('X_test_tfidf.shape=',X_test_tfidf.shape)

# MultinomialNB
from sklearn.naive_bayes import MultinomialNB
model_mnb_ROS = MultinomialNB()
#
model_mnb_ROS.fit(X_train_tfidf,y_train)
model_mnb_ROS.score(X_train_tfidf,y_train),model_mnb_ROS.score(X_test_tfidf
,y_test)
```

```
(0.9749167855444603, 0.9557932263814617)
```

```
# test for df_test et df_htest datasets
Xtest_h=X_test_for_htest['reviews_title_text']
model_mnb_ROS.score(tf_idf_ROS.transform(Xtest_h),df_htest['sentiment'])
```

```
0.8746238716148446
```

```
# Prediction
predict_test_h=model_mnb_ROS.predict(tf_idf_ROS.transform(Xtest_h))
predict_test_h_unique=set(predict_test_h)
predict_test_h_unique
```

```
{0, 1, 2}
```

```
pd.DataFrame(predict_test_h,columns=['sentiment']).value_counts()
```

```
sentiment
2            854
1            105
0             38
dtype: int64
```

P6. Metrics for evaluation

As we have class imbalance problem, we use those metrics for evaluating model performance: precision, recall, F1-score, and AUC-ROC curve. F1-Score metric will be used as a principal evaluation criteria for this project.

We proceed now, to evaluate BaggingClassifier with:

1- base_estimator=RandomForestClassifier()

2- base_estimator=XGBClassifier()

```
# With ROS RandomOverSampler
sentences,labels,X,X_test_for_htest,df_htest=init_data_treatment(i_txt_proc
ess=2,i_epoch=5,i_ROS_op=True)
```

```
# checking
len(X),len(sentences),len(labels)
```

```
(11217, 11217, 11217)
```

```
# Data preparation for fit()
y=pd.Series(labels)
if isinstance(X, pd.core.frame.DataFrame):
  X=X['reviews_title_text'] # X must be Series for train after
Xtest_h=X_test_for_htest['reviews_title_text']
ytest_h=df_htest['sentiment']
#
Xtest_h.shape,ytest_h.shape,y.shape
#
# Split() X.....
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, r
andom_state=101)
X_train.shape, X_test.shape, y_train.shape, y_test

# Tf-idf operation
from sklearn.feature_extraction.text import TfidfVectorizer
tf_idf = TfidfVectorizer()
#
# conversion of reviews in Tf-Idf score
X_train_tfidf = tf_idf.fit_transform(X_train)
X_test_tfidf=tf_idf.transform(X_test)
#print('X_test_tfidf.shape=',X_test_tfidf.shape)
```

Evaluate BaggingClassifier with base_estimator=RandomForestClassifier()

```
#from sklearn.tree import DecisionTreeClassifier
#from sklearn.ensemble import RandomForestClassifier

bag_model = BaggingClassifier(
  base_estimator=RandomForestClassifier(),
  n_estimators=100,
  max_samples=0.8,
  bootstrap=True,
  oob_score=True,
  random_state=0
  )
#
fit_and_evaluate_model(bag_model,X_train_tfidf,X_test_tfidf,y_train,y_test)
evaluate_model_data_h(bag_model,tf_idf_ROS,Xtest_h,ytest_h)
```

```
[[927   0   0]
 [  0 949   0]
 [  0   0 929]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       927
           1       1.00      1.00      1.00       949
           2       1.00      1.00      1.00       929

    accuracy                           1.00      2805
   macro avg       1.00      1.00      1.00      2805
weighted avg       1.00      1.00      1.00      2805

[[  7   0  17]
 [  0  12  27]
 [  0   0 934]]
              precision    recall  f1-score   support

           0       1.00      0.29      0.45        24
           1       1.00      0.31      0.47        39
           2       0.96      1.00      0.98       934

    accuracy                           0.96       997
   macro avg       0.99      0.53      0.63       997
weighted avg       0.96      0.96      0.94       997
```

Evaluate BaggingClassifier with base_estimator=XGBClassifier()

```
#from sklearn.tree import DecisionTreeClassifier
#from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier

bag_model = BaggingClassifier(
  base_estimator=XGBClassifier(),
  n_estimators=100,
  max_samples=0.8,
  bootstrap=True,
  oob_score=True,
  random_state=0
  )
#
fit_and_evaluate_model(bag_model,X_train_tfidf,X_test_tfidf,y_train,y_test)
evaluate_model_data_h(bag_model,tf_idf_ROS,Xtest_h,ytest_h)
```

```
[[927   0   0]
 [ 28 895  26]
 [ 23  72 834]]
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       927
           1       0.93      0.94      0.93       949
           2       0.97      0.90      0.93       929
```

```
    accuracy                           0.95      2805
   macro avg       0.95      0.95      0.95      2805
weighted avg       0.95      0.95      0.95      2805

[[ 15   3   6]
 [  5  24  10]
 [ 20  62 852]]
              precision    recall  f1-score   support

           0       0.38      0.62      0.47        24
           1       0.27      0.62      0.37        39
           2       0.98      0.91      0.95       934

    accuracy                           0.89       997
   macro avg       0.54      0.72      0.60       997
weighted avg       0.94      0.89      0.91       997
```

P7. Generation of models by Tree-based classifiers Random Forest and XGBoost

Case of Random Forest Classifier

Evaluation with class_weight option

```
# Without ROS RandomOverSampler
# ===> i_ROS_op=False
sentences,labels,X,X_test_for_htest,df_htest=init_data_treatment(i_txt_proc
ess=2,i_epoch=5,i_ROS_op=False)

# checking
len(X_test_for_htest),len(df_htest)

(997, 997)

# checking
len(X),len(sentences),len(labels)

(3990, 3990, 3990)

# Data preparation for fit()
y=pd.Series(labels)
if isinstance(X, pd.core.frame.DataFrame):
  X=X['reviews_title_text'] # X must be Series for train after
Xtest_h=X_test_for_htest['reviews_title_text']
ytest_h=df_htest['sentiment']
#
Xtest_h.shape,ytest_h.shape,y.shape
#
# Split() X.....
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, r
andom_state=101)
X_train.shape, X_test.shape, y_train.shape, y_test
```

```
# Tf-idf operation
from sklearn.feature_extraction.text import TfidfVectorizer
tf_idf = TfidfVectorizer()
#
# conversion of reviews in Tf-Idf score
X_train_tfidf = tf_idf.fit_transform(X_train)
X_test_tfidf=tf_idf.transform(X_test)
#print('X_test_tfidf.shape=',X_test_tfidf.shape)
```

```
# RandomForestClassifier model
from sklearn.ensemble import RandomForestClassifier
#
model_rfc = RandomForestClassifier(random_state=0, class_weight='balanced')
#
fit_and_evaluate_model(model_rfc,X_train_tfidf,X_test_tfidf,y_train,y_test)
evaluate_model_data_h(model_rfc,tf_idf,Xtest_h,ytest_h)
```

```
[[  3   0  19]
 [  0   6  33]
 [  0   0 937]]
              precision    recall  f1-score   support

           0       1.00      0.14      0.24        22
           1       1.00      0.15      0.27        39
           2       0.95      1.00      0.97       937

    accuracy                           0.95       998
   macro avg       0.98      0.43      0.49       998
weighted avg       0.95      0.95      0.93       998

[[  4   0  20]
 [  0   6  33]
 [  0   0 934]]
              precision    recall  f1-score   support

           0       1.00      0.17      0.29        24
           1       1.00      0.15      0.27        39
           2       0.95      1.00      0.97       934

    accuracy                           0.95       997
   macro avg       0.98      0.44      0.51       997
weighted avg       0.95      0.95      0.93       997
```

Evaluation with RandomOverSampler operation

```
# With ROS RandomOverSampler
sentences,labels,X,X_test_for_htest,df_htest=init_data_treatment(i_txt_proc
ess=2,i_epoch=5,i_ROS_op=True)
```

```
# checking
len(X_test_for_htest),len(df_htest)
```

```
(997, 997)
```

```
# checking
len(X),len(sentences),len(labels)
```

```
(11217, 11217, 11217)
```

```
# Data preparation for fit()
y=pd.Series(labels)
if isinstance(X, pd.core.frame.DataFrame):
  X=X['reviews_title_text'] # X must be Series for train after
Xtest_h=X_test_for_htest['reviews_title_text']
ytest_h=df_htest['sentiment']
#
Xtest_h.shape,ytest_h.shape,y.shape
#
# Split() X.....
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, r
andom_state=101)
X_train.shape, X_test.shape, y_train.shape, y_test

# Tf-idf operation
from sklearn.feature_extraction.text import TfidfVectorizer
tf_idf_ROS = TfidfVectorizer()
#
# conversion of reviews in Tf-Idf score
X_train_tfidf = tf_idf_ROS.fit_transform(X_train)
X_test_tfidf=tf_idf_ROS.transform(X_test)
#print('X_test_tfidf.shape=',X_test_tfidf.shape)

# RandomForestClassifier model
from sklearn.ensemble import RandomForestClassifier
#
model_rfc_ROS = RandomForestClassifier(random_state=0)
#
fit_and_evaluate_model(model_rfc_ROS,X_train_tfidf,X_test_tfidf,y_train,y_t
est)
evaluate_model_data_h(model_rfc_ROS,tf_idf_ROS,Xtest_h,ytest_h)
```

```
[[927   0   0]
 [  0 949   0]
 [  0   0 929]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       927
           1       1.00      1.00      1.00       949
           2       1.00      1.00      1.00       929

    accuracy                           1.00      2805
   macro avg       1.00      1.00      1.00      2805
weighted avg       1.00      1.00      1.00      2805

[[  7   0  17]
 [  0  10  29]
 [  0   0 934]]
              precision    recall  f1-score   support
```

```
           0       1.00      0.29      0.45        24
           1       1.00      0.26      0.41        39
           2       0.95      1.00      0.98       934

    accuracy                           0.95       997
   macro avg       0.98      0.52      0.61       997
weighted avg       0.96      0.95      0.94       997
```

Case of XGBoost

Evaluation with scale_pos_weight option

```
# Without ROS RandomOverSampler
# ===> i_ROS_op=False
sentences,labels,X,X_test_for_htest,df_htest=init_data_treatment(i_txt_proc
ess=2,i_epoch=5,i_ROS_op=False)

# checking
len(X),len(sentences),len(labels)

(3990, 3990, 3990)

# Data preparation for fit()
y=pd.Series(labels)
if isinstance(X, pd.core.frame.DataFrame):
  X=X['reviews_title_text'] # X must be Series for train after
Xtest_h=X_test_for_htest['reviews_title_text']
ytest_h=df_htest['sentiment']
#
Xtest_h.shape,ytest_h.shape,y.shape
#
# Split() X.....
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, r
andom_state=101)
X_train.shape, X_test.shape, y_train.shape, y_test

# Tf-idf operation
from sklearn.feature_extraction.text import TfidfVectorizer
tf_idf = TfidfVectorizer()
#
# conversion of reviews in Tf-Idf score
X_train_tfidf = tf_idf.fit_transform(X_train)
X_test_tfidf=tf_idf.transform(X_test)
#print('X_test_tfidf.shape=',X_test_tfidf.shape)

# calculation the value of scale_pos_weight
coeff=int(len(df[df['sentiment']=='Positive'])/len(df[df['sentiment']=='Neg
ative']))
coeff

40

# XGBoost
#
```

```
from xgboost import XGBClassifier

model_xgb1 = XGBClassifier(scale_pos_weight=coeff)  # coeff=40
#
fit_and_evaluate_model(model_xgb1,X_train_tfidf,X_test_tfidf,y_train,y_test
)
evaluate_model_data_h(model_xgb1,tf_idf,Xtest_h,ytest_h)

[[  3   2  17]
 [  0   4  35]
 [  0   5 932]]
              precision    recall  f1-score   support

           0       1.00      0.14      0.24        22
           1       0.36      0.10      0.16        39
           2       0.95      0.99      0.97       937

    accuracy                           0.94       998
   macro avg       0.77      0.41      0.46       998
weighted avg       0.93      0.94      0.92       998

[[  4   1  19]
 [  1   4  34]
 [  1   0 933]]
              precision    recall  f1-score   support

           0       0.67      0.17      0.27        24
           1       0.80      0.10      0.18        39
           2       0.95      1.00      0.97       934

    accuracy                           0.94       997
   macro avg       0.80      0.42      0.47       997
weighted avg       0.93      0.94      0.92       997
```

Evaluation with RandomOverSampler operation

```
# using function init_data_treatment()
# imblearn.over_sampling.RandomOverSampler to handle imbalanced data
# ===> i_ROS_op=True

sentences,labels,X,X_test_for_htest,df_htest=init_data_treatment(i_txt_proc
ess=2,i_epoch=5,i_ROS_op=True)

# checking
len(X),len(sentences),len(labels)

(11217, 11217, 11217)

# Data preparation for fit()
y=pd.Series(labels)
if isinstance(X, pd.core.frame.DataFrame):
  X=X['reviews_title_text'] # X must be Series for train after
Xtest_h=X_test_for_htest['reviews_title_text']
ytest_h=df_htest['sentiment']
```

```
#
Xtest_h.shape,ytest_h.shape,y.shape
#
# Split() X.....
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, r
andom_state=101)
X_train.shape, X_test.shape, y_train.shape, y_test

# Tf-idf operation
from sklearn.feature_extraction.text import TfidfVectorizer
tf_idf_ROS = TfidfVectorizer()
#
# conversion of reviews in Tf-Idf score
X_train_tfidf = tf_idf_ROS.fit_transform(X_train)
X_test_tfidf=tf_idf_ROS.transform(X_test)
#print('X_test_tfidf.shape=',X_test_tfidf.shape)

# XGBoost
#
from xgboost import XGBClassifier
model_xgb_ROS = XGBClassifier()
#
fit_and_evaluate_model(model_xgb_ROS,X_train_tfidf,X_test_tfidf,y_train,y_t
est)
evaluate_model_data_h(model_xgb_ROS,tf_idf_ROS,Xtest_h,ytest_h)
```

```
[[927   0   0]
 [ 28 895  26]
 [ 21  64 844]]
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       927
           1       0.93      0.94      0.94       949
           2       0.97      0.91      0.94       929

    accuracy                           0.95      2805
   macro avg       0.95      0.95      0.95      2805
weighted avg       0.95      0.95      0.95      2805

[[ 16   3   5]
 [  4  24  11]
 [ 19  57 858]]
              precision    recall  f1-score   support

           0       0.41      0.67      0.51        24
           1       0.29      0.62      0.39        39
           2       0.98      0.92      0.95       934

    accuracy                           0.90       997
   macro avg       0.56      0.73      0.62       997
weighted avg       0.94      0.90      0.92       997
```

III- Model Selection

P8. Generation of models by the two approaches: multi-class SVM’s and neural nets

Generation of models by multi-class SVM’s approch

Evaluation without RandomOverSampler operation

```
# Without ROS RandomOverSampler
# ===> i_ROS_op=False
sentences,labels,X,X_test_for_htest,df_htest=init_data_treatment(i_txt_proc
ess=2,i_epoch=5,i_ROS_op=False)

# checking
len(X),len(sentences),len(labels)

(3990, 3990, 3990)

# Data preparation for fit()
y=pd.Series(labels)
if isinstance(X, pd.core.frame.DataFrame):
  X=X['reviews_title_text'] # X must be Series for train after
Xtest_h=X_test_for_htest['reviews_title_text']
ytest_h=df_htest['sentiment']
#
Xtest_h.shape,ytest_h.shape,y.shape
#
# Split() X.....
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, r
andom_state=101)
X_train.shape, X_test.shape, y_train.shape, y_test

# Tf-idf operation
from sklearn.feature_extraction.text import TfidfVectorizer
tf_idf = TfidfVectorizer()
#
# conversion of reviews in Tf-Idf score
X_train_tfidf = tf_idf.fit_transform(X_train)
X_test_tfidf=tf_idf.transform(X_test)
#print('X_test_tfidf.shape=',X_test_tfidf.shape)

from sklearn.svm import SVC
model_svc = SVC()
#
fit_and_evaluate_model(model_svc,X_train_tfidf,X_test_tfidf,y_train,y_test)
evaluate_model_data_h(model_svc,tf_idf,Xtest_h,ytest_h)

[[   1    0   21]
 [   0    3   36]
 [   0    0 937]]
              precision    recall  f1-score   support
```

```
           0       1.00      0.05      0.09        22
           1       1.00      0.08      0.14        39
           2       0.94      1.00      0.97       937

    accuracy                           0.94       998
   macro avg       0.98      0.37      0.40       998
weighted avg       0.95      0.94      0.92       998

[[  4   0  20]
 [  0   4  35]
 [  0   0 934]]
              precision    recall  f1-score   support

           0       1.00      0.17      0.29        24
           1       1.00      0.10      0.19        39
           2       0.94      1.00      0.97       934

    accuracy                           0.94       997
   macro avg       0.98      0.42      0.48       997
weighted avg       0.95      0.94      0.92       997
```

Evaluation with RandomOverSampler operation

```
# using function init_data_treatment()
# imblearn.over_sampling.RandomOverSampler to handle imbalanced data
# ===> i_ROS_op=True

sentences,labels,X,X_test_for_htest,df_htest=init_data_treatment(i_txt_proc
ess=2,i_epoch=5,i_ROS_op=True)

# checking
len(X),len(sentences),len(labels)

(11217, 11217, 11217)

# Data preparation for fit()
y=pd.Series(labels)
if isinstance(X, pd.core.frame.DataFrame):
  X=X['reviews_title_text'] # X must be Series for train after
Xtest_h=X_test_for_htest['reviews_title_text']
ytest_h=df_htest['sentiment']
#
Xtest_h.shape,ytest_h.shape,y.shape
#
# Split() X.....
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, r
andom_state=101)
X_train.shape, X_test.shape, y_train.shape, y_test

# Tf-idf operation
from sklearn.feature_extraction.text import TfidfVectorizer
tf_idf_ROS = TfidfVectorizer()
```

```
#
# conversion of reviews in Tf-Idf score
X_train_tfidf = tf_idf_ROS.fit_transform(X_train)
X_test_tfidf=tf_idf_ROS.transform(X_test)
#print('X_test_tfidf.shape=',X_test_tfidf.shape)

from sklearn.svm import SVC
model_svc_ROS = SVC()
#
fit_and_evaluate_model(model_svc_ROS,X_train_tfidf,X_test_tfidf,y_train,y_t
est)
evaluate_model_data_h(model_svc_ROS,tf_idf_ROS,Xtest_h,ytest_h)
```

```
[[927   0   0]
 [  0 949   0]
 [  0   1 928]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       927
           1       1.00      1.00      1.00       949
           2       1.00      1.00      1.00       929

    accuracy                           1.00      2805
   macro avg       1.00      1.00      1.00      2805
weighted avg       1.00      1.00      1.00      2805

[[  8   0  16]
 [  0  11  28]
 [  0   0 934]]
              precision    recall  f1-score   support

           0       1.00      0.33      0.50        24
           1       1.00      0.28      0.44        39
           2       0.96      1.00      0.98       934

    accuracy                           0.96       997
   macro avg       0.99      0.54      0.64       997
weighted avg       0.96      0.96      0.94       997
```

Generation of models by neural nets approch

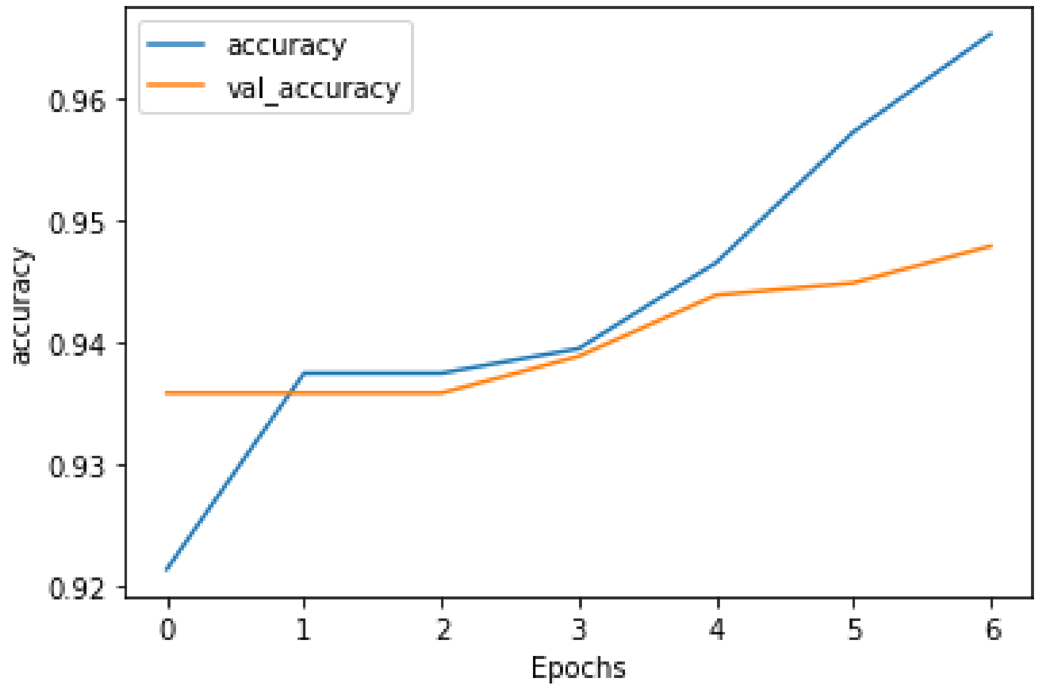
Evaluation without RandomOverSampler operation

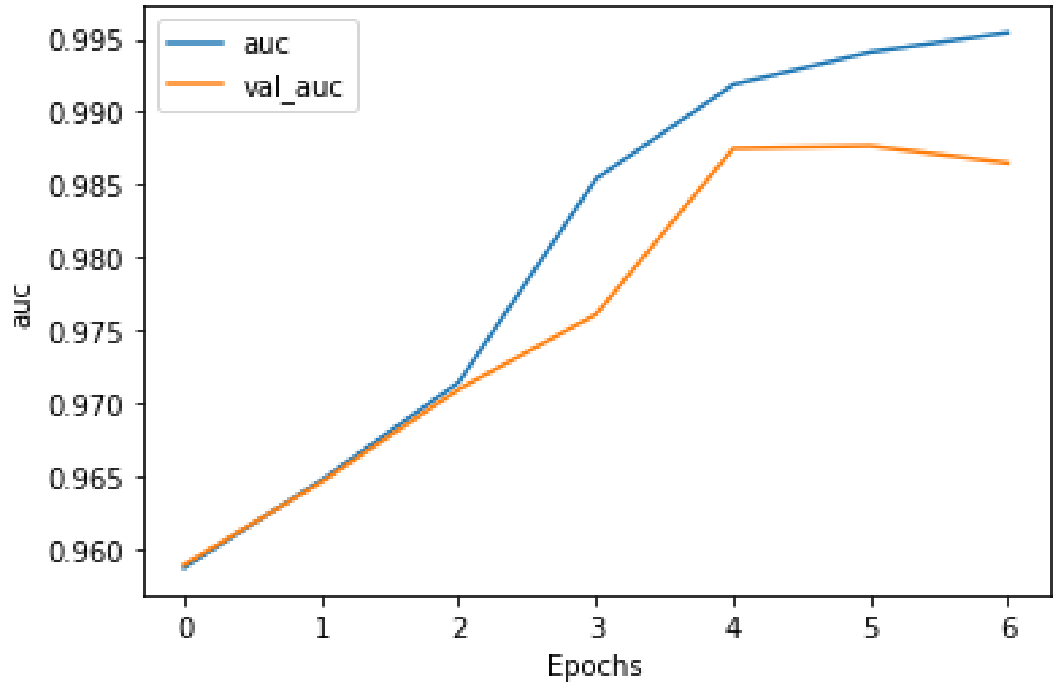
```
# Without ROS RandomOverSampler
# ===> i_ROS_op=False
sentences,labels,X,X_test_for_htest,df_htest=init_data_treatment(i_txt_proc
ess=2,i_epoch=7,i_ROS_op=False)

X.shape

(3990, 1)
```

```
################################
#*** Treatment---> Case 1_layer_Dense
###############################

#'Bi_LTSM', 1_layer_GRU, 1_layer_LTSM
choise_model='1_layer_Dense'

# model generation
(model_tf,history_tf)=treatment_case_tensorflow(sentences,labels,choise_mod
el,tokenizer,dico_params)

# plot history graphs history for metrics accuracy and AUC
plot_graphs_history(history_tf, 'accuracy')
plot_graphs_history(history_tf, 'auc')
#plot_graphs_history(history_tf, 'loss')

# evaluation
evaluate_model_tf(model_tf,X_test_for_htest,df_htest,dico_params,tokenizer)
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 120, 16)           160000

 dropout_6 (Dropout)         (None, 120, 16)           0

 dense_6 (Dense)             (None, 120, 30)           510

 dropout_7 (Dropout)         (None, 120, 30)           0

 flatten_3 (Flatten)         (None, 3600)              0

 dense_7 (Dense)             (None, 3)                 10803

 activation_3 (Activation)   (None, 3)                 0

=================================================================
Total params: 171,313
Trainable params: 171,313
Non-trainable params: 0
_________________________________________________________________
Epoch 1/7
94/94 [==============================] - 3s 15ms/step - loss: 0.2259 - accu
racy: 0.9215 - auc: 0.9587 - val_loss: 0.1683 - val_accuracy: 0.9359 - val_
auc: 0.9590
Epoch 2/7
94/94 [==============================] - 1s 10ms/step - loss: 0.1626 - accu
racy: 0.9375 - auc: 0.9647 - val_loss: 0.1624 - val_accuracy: 0.9359 - val_
auc: 0.9646
Epoch 3/7
94/94 [==============================] - 1s 11ms/step - loss: 0.1489 - accu
racy: 0.9375 - auc: 0.9715 - val_loss: 0.1490 - val_accuracy: 0.9359 - val_
auc: 0.9710
```

```
Epoch 4/7
94/94 [==============================] - 1s 13ms/step - loss: 0.1244 - accu
racy: 0.9395 - auc: 0.9854 - val_loss: 0.1308 - val_accuracy: 0.9389 - val_
auc: 0.9761
Epoch 5/7
94/94 [==============================] - 1s 8ms/step - loss: 0.0943 - accur
acy: 0.9465 - auc: 0.9919 - val_loss: 0.1077 - val_accuracy: 0.9439 - val_a
uc: 0.9875
Epoch 6/7
94/94 [==============================] - 1s 7ms/step - loss: 0.0760 - accur
acy: 0.9572 - auc: 0.9941 - val_loss: 0.1029 - val_accuracy: 0.9449 - val_a
uc: 0.9877
Epoch 7/7
94/94 [==============================] - 1s 7ms/step - loss: 0.0634 - accur
acy: 0.9652 - auc: 0.9954 - val_loss: 0.1031 - val_accuracy: 0.9479 - val_a
uc: 0.9865
```






```
[[  9   2  13]
 [  2   8  29]
 [  0   3 931]]
              precision    recall  f1-score   support

           0       0.82      0.38      0.51        24
           1       0.62      0.21      0.31        39
           2       0.96      1.00      0.98       934

    accuracy                           0.95       997
   macro avg       0.80      0.53      0.60       997
weighted avg       0.94      0.95      0.94       997
```

Evaluation with RandomOverSampler operation

```
# With ROS RandomOverSampler
# ===> i_ROS_op=True
sentences,labels,X,X_test_for_htest,df_htest=init_data_treatment(i_txt_proc
ess=2,i_epoch=7,i_ROS_op=True)
```

```
X.shape
```

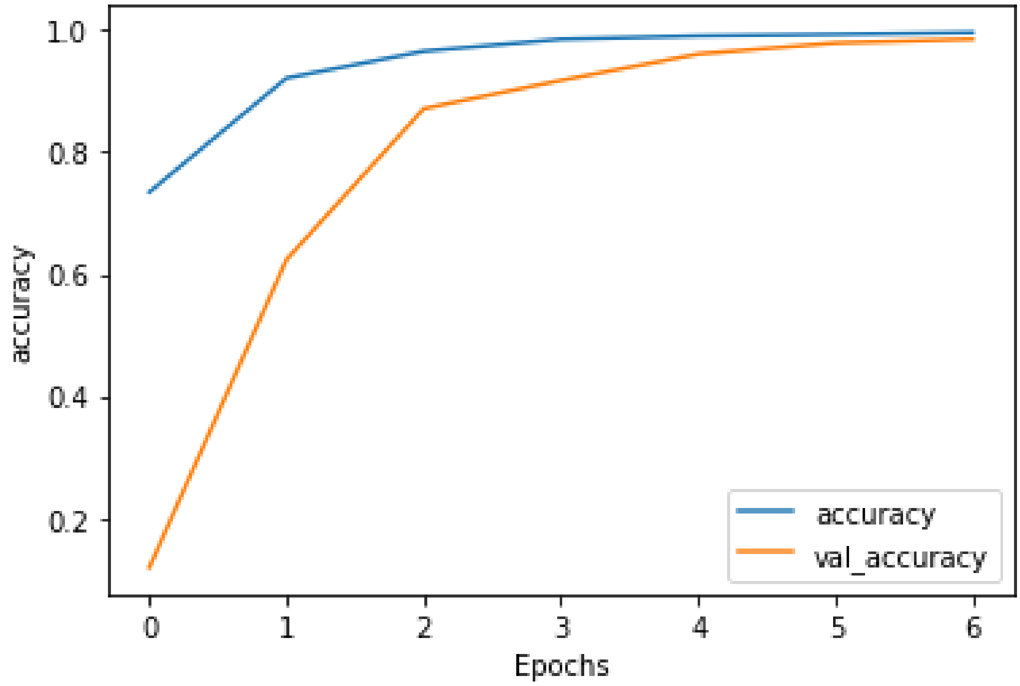
```
(11217, 1)
```

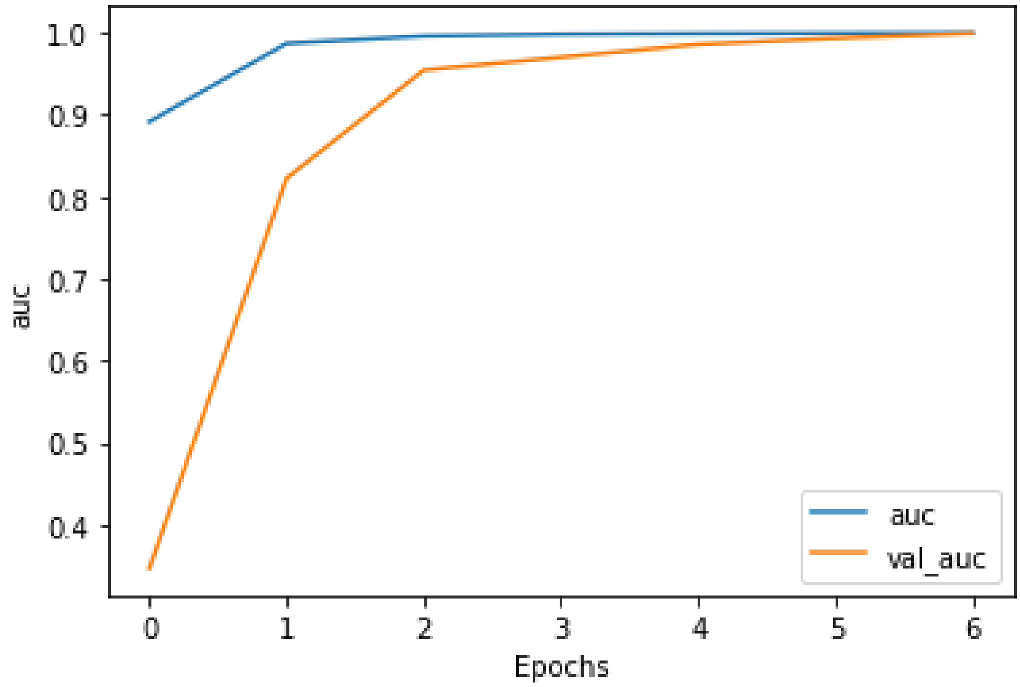
```
################################
#*** Treatment---> Case 1_layer_Dense
###############################

#'Bi_LTSM', 1_layer_GRU, 1_layer_LTSM
choise_model='1_layer_Dense'

# model generation
```

```
(model_tf,history_tf)=treatment_case_tensorflow(sentences,labels,choise_mod
el,tokenizer,dico_params)

# plot history graphs history for metrics accuracy and AUC
plot_graphs_history(history_tf, 'accuracy')
plot_graphs_history(history_tf, 'auc')
#plot_graphs_history(history_tf, 'loss')

# evaluation
evaluate_model_tf(model_tf,X_test_for_htest,df_htest,dico_params,tokenizer)
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_8 (Embedding)     (None, 120, 16)           160000    
                                                                 
 dropout_16 (Dropout)        (None, 120, 16)           0         
                                                                 
 dense_16 (Dense)            (None, 120, 30)           510       
                                                                 
 dropout_17 (Dropout)        (None, 120, 30)           0         
                                                                 
 flatten_8 (Flatten)         (None, 3600)              0         
                                                                 
 dense_17 (Dense)            (None, 3)                 10803     
                                                                 
 activation_8 (Activation)   (None, 3)                 0         
                                                                 
=================================================================
Total params: 171,313
Trainable params: 171,313
Non-trainable params: 0
_________________________________________________________________
Epoch 1/7
263/263 [==============================] - 3s 8ms/step - loss: 0.4118 - acc
uracy: 0.7345 - auc: 0.8915 - val_loss: 0.9607 - val_accuracy: 0.1191 - val
_auc: 0.3473
Epoch 2/7
263/263 [==============================] - 2s 7ms/step - loss: 0.1473 - acc
uracy: 0.9214 - auc: 0.9869 - val_loss: 0.5092 - val_accuracy: 0.6239 - val
_auc: 0.8226
Epoch 3/7
263/263 [==============================] - 2s 7ms/step - loss: 0.0812 - acc
uracy: 0.9654 - auc: 0.9958 - val_loss: 0.2653 - val_accuracy: 0.8713 - val
_auc: 0.9543
Epoch 4/7
263/263 [==============================] - 2s 7ms/step - loss: 0.0458 - acc
uracy: 0.9850 - auc: 0.9980 - val_loss: 0.1819 - val_accuracy: 0.9176 - val
_auc: 0.9700
Epoch 5/7
263/263 [==============================] - 2s 7ms/step - loss: 0.0308 - acc
uracy: 0.9906 - auc: 0.9988 - val_loss: 0.0985 - val_accuracy: 0.9608 - val
_auc: 0.9857
```

```
Epoch 6/7
263/263 [==============================] - 2s 7ms/step - loss: 0.0223 - acc
uracy: 0.9931 - auc: 0.9992 - val_loss: 0.0560 - val_accuracy: 0.9790 - val
_auc: 0.9932
Epoch 7/7
263/263 [==============================] - 2s 7ms/step - loss: 0.0154 - acc
uracy: 0.9962 - auc: 0.9995 - val_loss: 0.0392 - val_accuracy: 0.9850 - val
_auc: 0.9992
```






```
[[  9   3  12]
 [  3  16  20]
```

```
 [  2   6 926]]
              precision    recall  f1-score   support

           0       0.64      0.38      0.47        24
           1       0.64      0.41      0.50        39
           2       0.97      0.99      0.98       934

    accuracy                           0.95       997
   macro avg       0.75      0.59      0.65       997
weighted avg       0.95      0.95      0.95       997
```

P9. Generation of models by ensemble techniques: oversampled XGboost and oversampled multinomial_NB

```
# imblearn.over_sampling.RandomOverSampler to handle imbalanced data
# ===> i_ROS_op=True

sentences,labels,X,X_test_for_htest,df_htest=init_data_treatment(i_txt_proc
ess=2,i_epoch=5,i_ROS_op=True)

# checking
len(X),len(sentences),len(labels)

(11217, 11217, 11217)

# Data preparation for fit()
y=pd.Series(labels)
if isinstance(X, pd.core.frame.DataFrame):
  X=X['reviews_title_text'] # X must be Series for train after
Xtest_h=X_test_for_htest['reviews_title_text']
ytest_h=df_htest['sentiment']
#
Xtest_h.shape,ytest_h.shape,y.shape
#
# Split() X.....
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, r
andom_state=101)
X_train.shape, X_test.shape, y_train.shape, y_test

# Tf-idf operation
from sklearn.feature_extraction.text import TfidfVectorizer
tf_idf_ROS = TfidfVectorizer()
#
# conversion of reviews in Tf-Idf score
X_train_tfidf = tf_idf_ROS.fit_transform(X_train)
X_test_tfidf=tf_idf_ROS.transform(X_test)
#print('X_test_tfidf.shape=',X_test_tfidf.shape)
```

###Case XGBoost

```
# XGBoost
#
from xgboost import XGBClassifier
```

```
model_xgb_ROS = XGBClassifier(
  n_splits=10,
  learning_rate =0.1,
  n_estimators=1000,
  max_depth=5,
  min_child_weight=1,
  gamma=0,
  subsample=0.8,
  colsample_bytree=0.8,
  objective= 'binary:logistic',
  scale_pos_weight=1,
  seed=27)

fit_and_evaluate_model(model_xgb_ROS,X_train_tfidf,X_test_tfidf,y_train,y_t
est)
evaluate_model_data_h(model_xgb_ROS,tf_idf_ROS,Xtest_h,ytest_h)

[[927   0   0]
 [  0 949   0]
 [  3  12 914]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       927
           1       0.99      1.00      0.99       949
           2       1.00      0.98      0.99       929

    accuracy                           0.99      2805
   macro avg       0.99      0.99      0.99      2805
weighted avg       0.99      0.99      0.99      2805

[[ 13   2   9]
 [  2  14  23]
 [  2   8 924]]
              precision    recall  f1-score   support

           0       0.76      0.54      0.63        24
           1       0.58      0.36      0.44        39
           2       0.97      0.99      0.98       934

    accuracy                           0.95       997
   macro avg       0.77      0.63      0.69       997
weighted avg       0.95      0.95      0.95       997


# Another test for XGBClassifier()
'''
model_xgb_ROS = XGBClassifier(n_splits=10,base_score=0.5, booster='gbtree',
colsample_bylevel=1,
colsample_bynode=1, colsample_bytree=0.9, gamma=0,
learning_rate=0.1, max_delta_step=0, max_depth=10,
min_child_weight=1, missing=None, n_estimators=500, n_jobs=-1,
nthread=None, objective='binary:logistic', random_state=0,
reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
```

```
silent=None, subsample=0.9, verbosity=0)
#
fit_and_evaluate_model(model_xgb_ROS,X_train_tfidf,X_test_tfidf,y_train,y_t
est)
evaluate_model_data_h(model_xgb_ROS,tf_idf_ROS,Xtest_h,ytest_h)
'''
```

Case Naive Bayes: MultinomialNB

```
# MultinomialNB
from sklearn.naive_bayes import MultinomialNB
model_mnb_ROS = MultinomialNB(alpha=0.7,fit_prior=True)
#
fit_and_evaluate_model(model_mnb_ROS,X_train_tfidf,X_test_tfidf,y_train,y_t
est)
evaluate_model_data_h(model_mnb_ROS,tf_idf_ROS,Xtest_h,ytest_h)
```

```
[[927   0   0]
 [  0 934  15]
 [ 20  76 833]]
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       927
           1       0.92      0.98      0.95       949
           2       0.98      0.90      0.94       929

    accuracy                           0.96      2805
   macro avg       0.96      0.96      0.96      2805
weighted avg       0.96      0.96      0.96      2805

[[ 11   7   6]
 [  4  20  15]
 [ 19  69 846]]
              precision    recall  f1-score   support

           0       0.32      0.46      0.38        24
           1       0.21      0.51      0.30        39
           2       0.98      0.91      0.94       934

    accuracy                           0.88       997
   macro avg       0.50      0.63      0.54       997
weighted avg       0.93      0.88      0.90       997
```

IV- Neural network models: Application of LSTM and GRU layers

P11. GRU Layers

Evaluation without RandomOverSampler operation

```
# Without ROS RandomOverSampler
# ===> i_ROS_op=False
sentences,labels,X,X_test_for_htest,df_htest=init_data_treatment(i_txt_proc
ess=2,i_epoch=12,i_ROS_op=False)
```

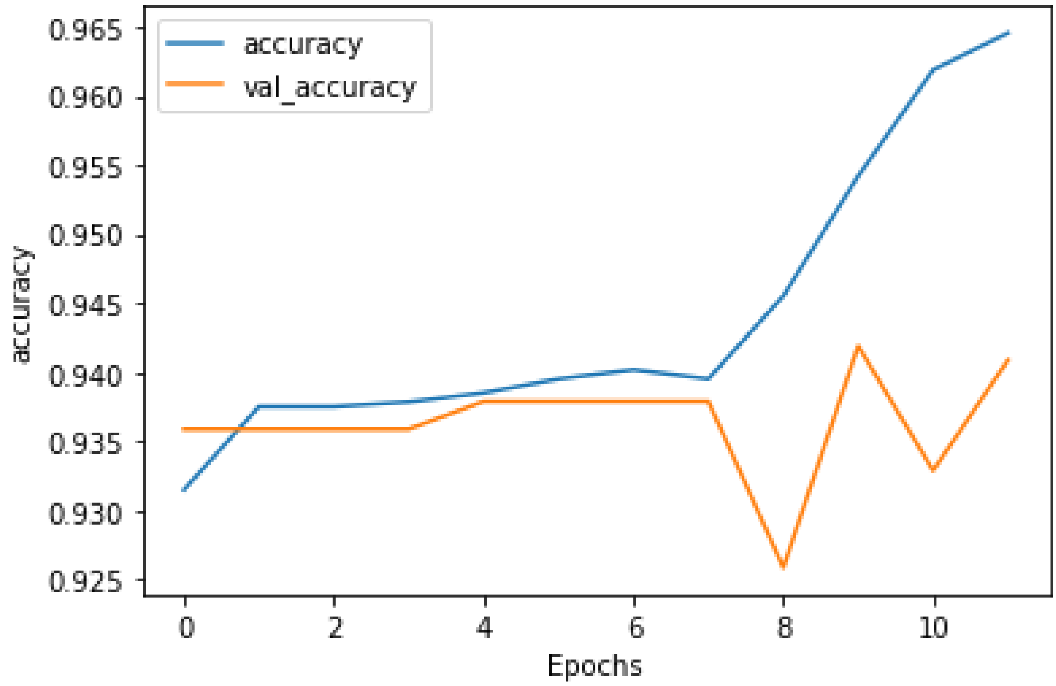
```
X.shape
```

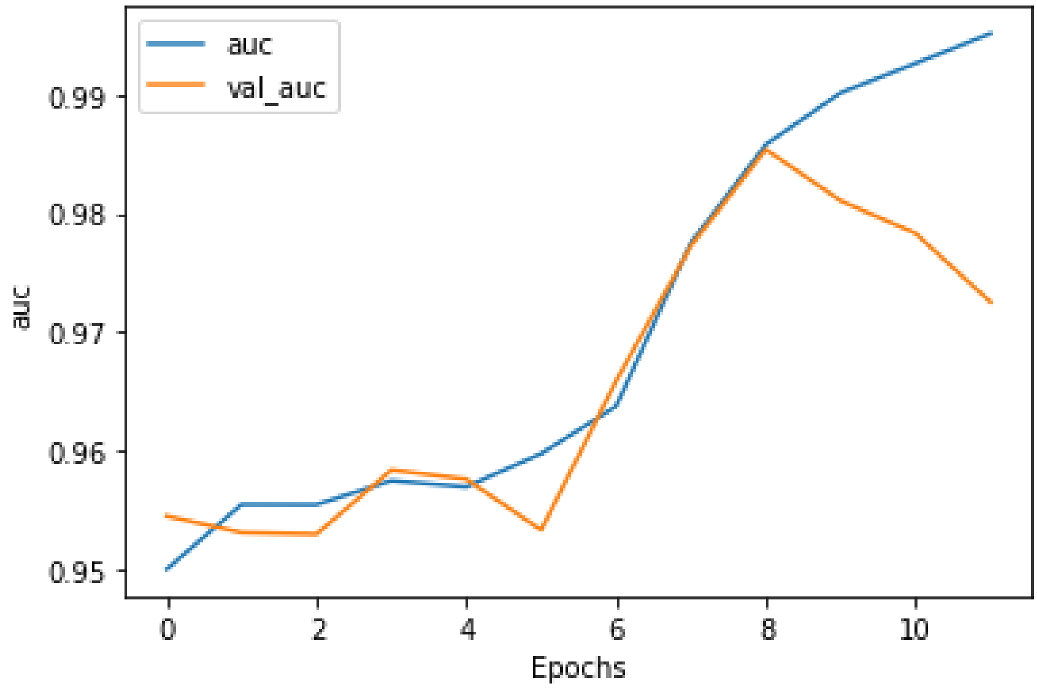
```
(3990, 1)
```

```
################################
#*** Treatment---> Case 1_layer_GRU
###############################

#'Bi_LTSM', 1_layer_GRU, 1_layer_LTSM
choise_model='1_layer_GRU'

# model generation
(model_tf,history_tf)=treatment_case_tensorflow(sentences,labels,choise_mod
el,tokenizer,dico_params)

# plot history graphs history for metrics accuracy and AUC
plot_graphs_history(history_tf, 'accuracy')
plot_graphs_history(history_tf, 'auc')
#plot_graphs_history(history_tf, 'loss')

# evaluation
evaluate_model_tf(model_tf,X_test_for_htest,df_htest,dico_params,tokenizer)
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_10 (Embedding)    (None, 120, 16)           160000

 dropout_20 (Dropout)        (None, 120, 16)           0

 gru_1 (GRU)                 (None, 120)               49680

 dropout_21 (Dropout)        (None, 120)               0

 dense_19 (Dense)            (None, 3)                 363

 activation_10 (Activation)  (None, 3)                 0

=================================================================
Total params: 210,043
Trainable params: 210,043
Non-trainable params: 0
_________________________________________________________________
Epoch 1/12
94/94 [==============================] - 11s 92ms/step - loss: 0.2723 - acc
uracy: 0.9315 - auc: 0.9500 - val_loss: 0.1723 - val_accuracy: 0.9359 - val
_auc: 0.9545
Epoch 2/12
94/94 [==============================] - 8s 90ms/step - loss: 0.1796 - accu
```

```
racy: 0.9375 - auc: 0.9555 - val_loss: 0.1752 - val_accuracy: 0.9359 - val_
auc: 0.9531
Epoch 3/12
94/94 [==============================] - 8s 87ms/step - loss: 0.1763 - accu
racy: 0.9375 - auc: 0.9554 - val_loss: 0.1734 - val_accuracy: 0.9359 - val_
auc: 0.9529
Epoch 4/12
94/94 [==============================] - 8s 89ms/step - loss: 0.1734 - accu
racy: 0.9378 - auc: 0.9574 - val_loss: 0.1698 - val_accuracy: 0.9359 - val_
auc: 0.9583
Epoch 5/12
94/94 [==============================] - 8s 87ms/step - loss: 0.1697 - accu
racy: 0.9385 - auc: 0.9569 - val_loss: 0.1683 - val_accuracy: 0.9379 - val_
auc: 0.9576
Epoch 6/12
94/94 [==============================] - 13s 138ms/step - loss: 0.1666 - ac
curacy: 0.9395 - auc: 0.9597 - val_loss: 0.1703 - val_accuracy: 0.9379 - va
l_auc: 0.9533
Epoch 7/12
94/94 [==============================] - 8s 87ms/step - loss: 0.1606 - accu
racy: 0.9402 - auc: 0.9637 - val_loss: 0.1579 - val_accuracy: 0.9379 - val_
auc: 0.9659
Epoch 8/12
94/94 [==============================] - 8s 86ms/step - loss: 0.1367 - accu
racy: 0.9395 - auc: 0.9775 - val_loss: 0.1318 - val_accuracy: 0.9379 - val_
auc: 0.9773
Epoch 9/12
94/94 [==============================] - 8s 86ms/step - loss: 0.1045 - accu
racy: 0.9455 - auc: 0.9858 - val_loss: 0.1376 - val_accuracy: 0.9259 - val_
auc: 0.9854
Epoch 10/12
94/94 [==============================] - 8s 88ms/step - loss: 0.0844 - accu
racy: 0.9542 - auc: 0.9902 - val_loss: 0.1199 - val_accuracy: 0.9419 - val_
auc: 0.9811
Epoch 11/12
94/94 [==============================] - 15s 163ms/step - loss: 0.0699 - ac
curacy: 0.9619 - auc: 0.9927 - val_loss: 0.1289 - val_accuracy: 0.9329 - va
l_auc: 0.9783
Epoch 12/12
94/94 [==============================] - 14s 150ms/step - loss: 0.0573 - ac
curacy: 0.9646 - auc: 0.9952 - val_loss: 0.1519 - val_accuracy: 0.9409 - va
l_auc: 0.9725
```






```
[[  2  10  12]
 [  0  15  24]
 [  2   8 924]]
              precision    recall  f1-score   support

           0       0.50      0.08      0.14        24
           1       0.45      0.38      0.42        39
           2       0.96      0.99      0.98       934

    accuracy                           0.94       997
   macro avg       0.64      0.49      0.51       997
```

```
weighted avg       0.93      0.94      0.93       997
```

Evaluation with RandomOverSampler operation

```
# Without ROS RandomOverSampler
# ===> i_ROS_op=True
sentences,labels,X,X_test_for_htest,df_htest=init_data_treatment(i_txt_proc
ess=2,i_epoch=12,i_ROS_op=True)
```

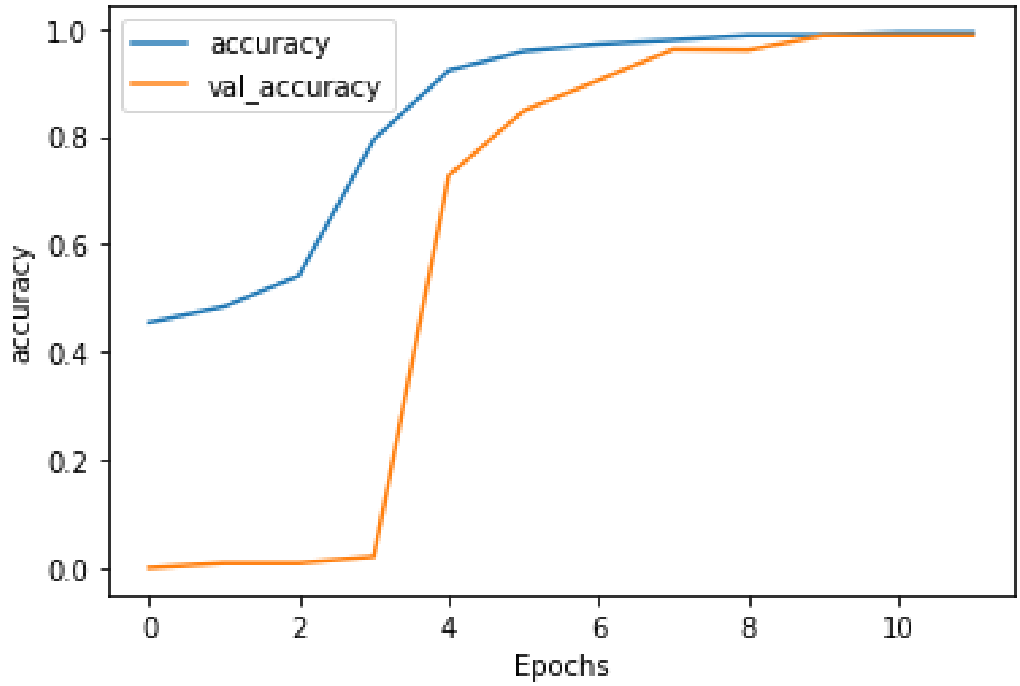
```
X.shape
```

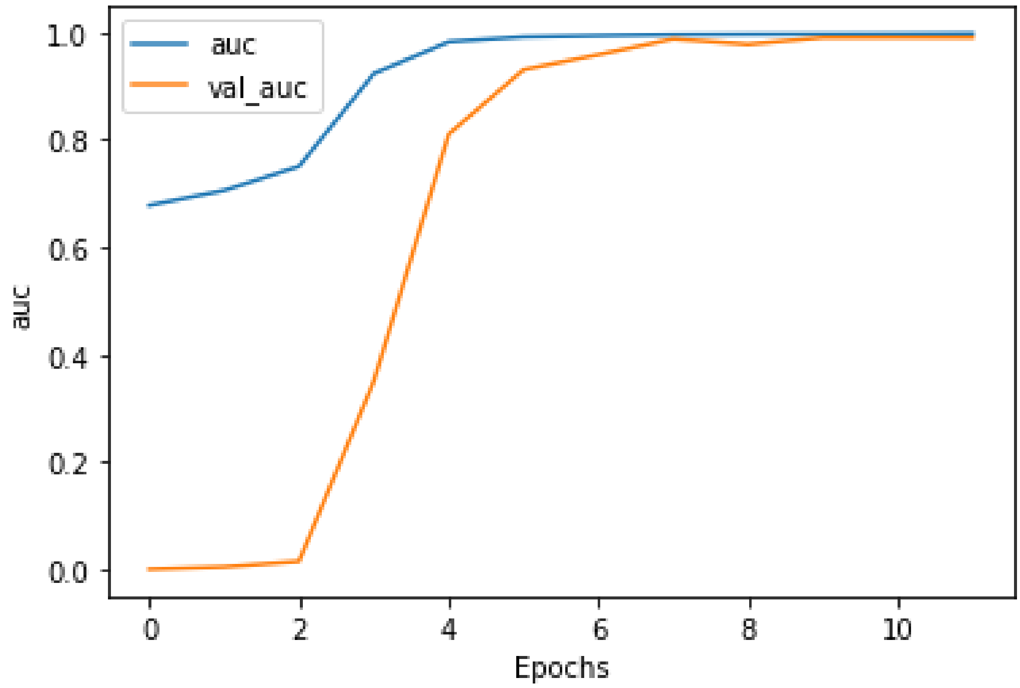
```
(11217, 1)
```

```
################################
#*** Treatment---> Case 1_layer_GRU
###############################

#'Bi_LTSM', 1_layer_GRU, 1_layer_LTSM
choise_model='1_layer_GRU'

# model generation
(model_tf,history_tf)=treatment_case_tensorflow(sentences,labels,choise_mod
el,tokenizer,dico_params)

# plot history graphs history for metrics accuracy and AUC
plot_graphs_history(history_tf, 'accuracy')
plot_graphs_history(history_tf, 'auc')
#plot_graphs_history(history_tf, 'loss')

# evaluation
evaluate_model_tf(model_tf,X_test_for_htest,df_htest,dico_params,tokenizer)
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_11 (Embedding)    (None, 120, 16)           160000    
                                                                 
 dropout_22 (Dropout)        (None, 120, 16)           0         
                                                                 
 gru_2 (GRU)                 (None, 120)               49680     
                                                                 
 dropout_23 (Dropout)        (None, 120)               0         
                                                                 
 dense_20 (Dense)            (None, 3)                 363       
                                                                 
 activation_11 (Activation)  (None, 3)                 0         
                                                                 
=================================================================
Total params: 210,043
Trainable params: 210,043
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 1/12
263/263 [==============================] - 26s 90ms/step - loss: 0.5817 - a
ccuracy: 0.4547 - auc: 0.6777 - val_loss: 1.0927 - val_accuracy: 0.0000e+00
- val_auc: 0.0000e+00
Epoch 2/12
263/263 [==============================] - 36s 138ms/step - loss: 0.5577 -
accuracy: 0.4842 - auc: 0.7054 - val_loss: 1.1420 - val_accuracy: 0.0086 -
val_auc: 0.0043
Epoch 3/12
263/263 [==============================] - 35s 132ms/step - loss: 0.5274 -
accuracy: 0.5416 - auc: 0.7502 - val_loss: 1.0830 - val_accuracy: 0.0086 -
val_auc: 0.0146
Epoch 4/12
263/263 [==============================] - 28s 106ms/step - loss: 0.3229 -
accuracy: 0.7941 - auc: 0.9230 - val_loss: 0.9788 - val_accuracy: 0.0200 -
val_auc: 0.3501
Epoch 5/12
263/263 [==============================] - 33s 126ms/step - loss: 0.1406 -
accuracy: 0.9221 - auc: 0.9822 - val_loss: 0.5342 - val_accuracy: 0.7273 -
val_auc: 0.8103
Epoch 6/12
263/263 [==============================] - 39s 147ms/step - loss: 0.0831 -
accuracy: 0.9580 - auc: 0.9914 - val_loss: 0.3071 - val_accuracy: 0.8474 -
val_auc: 0.9302
Epoch 7/12
263/263 [==============================] - 39s 148ms/step - loss: 0.0605 -
accuracy: 0.9711 - auc: 0.9937 - val_loss: 0.2393 - val_accuracy: 0.9041 -
val_auc: 0.9583
Epoch 8/12
263/263 [==============================] - 35s 133ms/step - loss: 0.0460 -
accuracy: 0.9787 - auc: 0.9956 - val_loss: 0.1223 - val_accuracy: 0.9615 -
val_auc: 0.9875
Epoch 9/12
263/263 [==============================] - 34s 129ms/step - loss: 0.0324 -
accuracy: 0.9873 - auc: 0.9963 - val_loss: 0.1637 - val_accuracy: 0.9601 -
val_auc: 0.9775
Epoch 10/12
263/263 [==============================] - 26s 97ms/step - loss: 0.0302 - a
ccuracy: 0.9878 - auc: 0.9967 - val_loss: 0.0710 - val_accuracy: 0.9872 - v
al_auc: 0.9900
Epoch 11/12
263/263 [==============================] - 33s 127ms/step - loss: 0.0227 -
accuracy: 0.9920 - auc: 0.9970 - val_loss: 0.0689 - val_accuracy: 0.9872 -
val_auc: 0.9903
Epoch 12/12
263/263 [==============================] - 28s 105ms/step - loss: 0.0199 -
accuracy: 0.9923 - auc: 0.9977 - val_loss: 0.0706 - val_accuracy: 0.9872 -
val_auc: 0.9902
```






```
[[ 11   4   9]
 [  3  14  22]
 [  0  25 909]]
              precision    recall  f1-score   support

           0       0.79      0.46      0.58        24
           1       0.33      0.36      0.34        39
           2       0.97      0.97      0.97       934

    accuracy                           0.94       997
   macro avg       0.69      0.60      0.63       997
```

```
weighted avg       0.94      0.94      0.94       997
```

P12. LSTM Layers

Simple LSTM Layers

Evaluation without RandomOverSampler operation

```
# Without ROS RandomOverSampler
# ===> i_ROS_op=False
sentences,labels,X,X_test_for_htest,df_htest=init_data_treatment(i_txt_proc
ess=2,i_epoch=7,i_ROS_op=False)
```

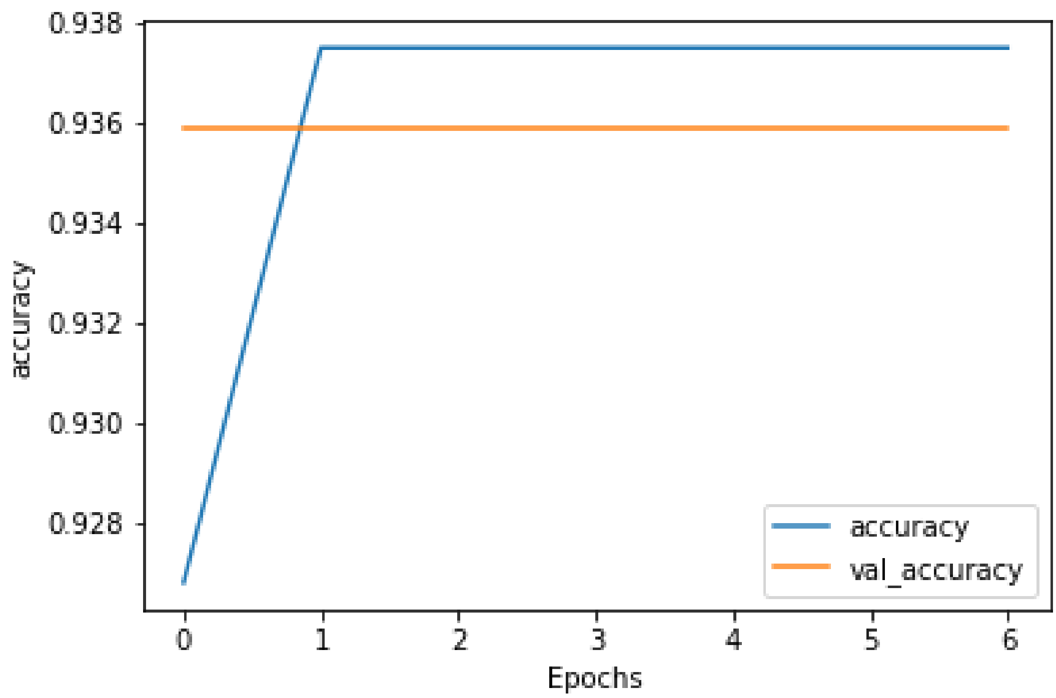
```
X.shape
```

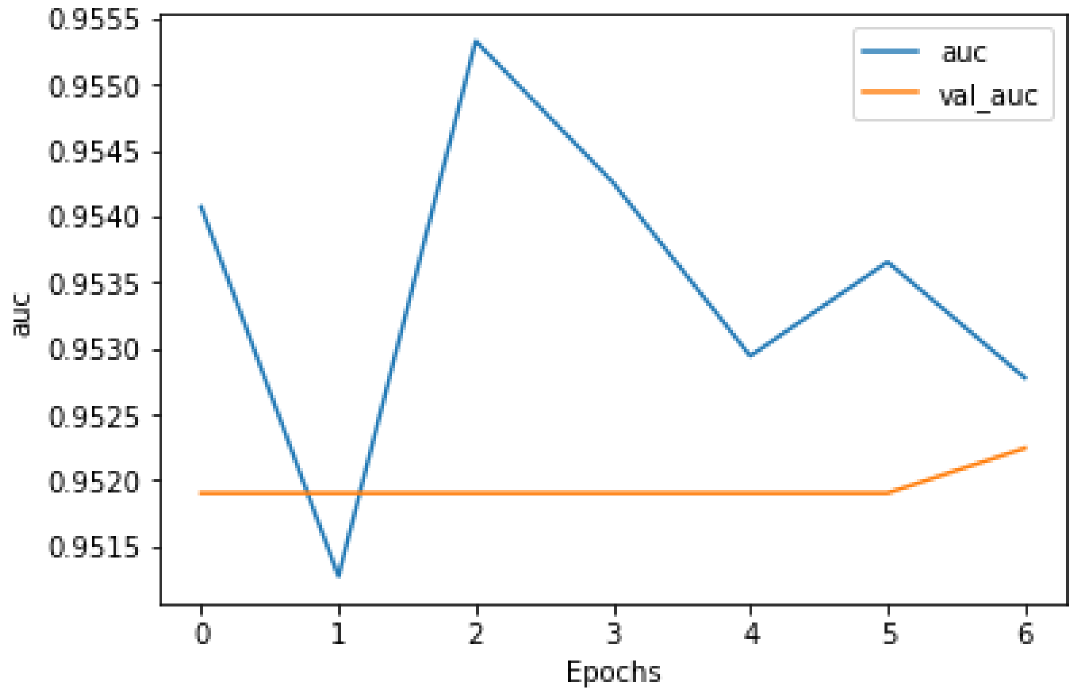
```
(3990, 1)
```

```
################################
#*** Treatment---> Case 1_layer_LSTM
###############################

#'Bi_LSTM', 1_layer_GRU, 1_layer_LSTM
choise_model='1_layer_LSTM'

# model generation
(model_tf,history_tf)=treatment_case_tensorflow(sentences,labels,choise_mod
el,tokenizer,dico_params)

# plot history graphs history for metrics accuracy and AUC
plot_graphs_history(history_tf, 'accuracy')
plot_graphs_history(history_tf, 'auc')
#plot_graphs_history(history_tf, 'loss')

# evaluation
evaluate_model_tf(model_tf,X_test_for_htest,df_htest,dico_params,tokenizer)
```

```
Model: "sequential_21"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_21 (Embedding)    (None, 120, 16)           160000    
                                                                 
 dropout_42 (Dropout)        (None, 120, 16)           0         
                                                                 
 lstm_9 (LSTM)               (None, 64)                20736     
                                                                 
 dropout_43 (Dropout)        (None, 64)                0         
                                                                 
 dense_30 (Dense)            (None, 3)                 195       
                                                                 
 activation_21 (Activation)  (None, 3)                 0         
                                                                 
=================================================================
```

```
Total params: 180,931
Trainable params: 180,931
Non-trainable params: 0
_________________________________________________________________
Epoch 1/7
94/94 [==============================] - 9s 74ms/step - loss: 0.2687 - accu
racy: 0.9268 - auc: 0.9541 - val_loss: 0.1742 - val_accuracy: 0.9359 - val_
auc: 0.9519
Epoch 2/7
94/94 [==============================] - 6s 68ms/step - loss: 0.1893 - accu
racy: 0.9375 - auc: 0.9513 - val_loss: 0.1746 - val_accuracy: 0.9359 - val_
auc: 0.9519
Epoch 3/7
94/94 [==============================] - 6s 68ms/step - loss: 0.1820 - accu
racy: 0.9375 - auc: 0.9553 - val_loss: 0.1751 - val_accuracy: 0.9359 - val_
auc: 0.9519
Epoch 4/7
94/94 [==============================] - 7s 75ms/step - loss: 0.1824 - accu
racy: 0.9375 - auc: 0.9543 - val_loss: 0.1734 - val_accuracy: 0.9359 - val_
auc: 0.9519
Epoch 5/7
94/94 [==============================] - 6s 69ms/step - loss: 0.1812 - accu
racy: 0.9375 - auc: 0.9529 - val_loss: 0.1738 - val_accuracy: 0.9359 - val_
auc: 0.9519
Epoch 6/7
94/94 [==============================] - 6s 69ms/step - loss: 0.1795 - accu
racy: 0.9375 - auc: 0.9537 - val_loss: 0.1731 - val_accuracy: 0.9359 - val_
auc: 0.9519
Epoch 7/7
94/94 [==============================] - 6s 69ms/step - loss: 0.1818 - accu
racy: 0.9375 - auc: 0.9528 - val_loss: 0.1729 - val_accuracy: 0.9359 - val_
auc: 0.9522
```






```
[[  0   0  24]
 [  0   0  39]
 [  0   0 934]]
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        24
           1       0.00      0.00      0.00        39
           2       0.94      1.00      0.97       934

    accuracy                           0.94       997
   macro avg       0.31      0.33      0.32       997
weighted avg       0.88      0.94      0.91       997

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1
318: UndefinedMetricWarning: Precision and F-score are ill-defined and bein
g set to 0.0 in labels with no predicted samples. Use `zero_division` param
eter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1
318: UndefinedMetricWarning: Precision and F-score are ill-defined and bein
g set to 0.0 in labels with no predicted samples. Use `zero_division` param
eter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1
318: UndefinedMetricWarning: Precision and F-score are ill-defined and bein
g set to 0.0 in labels with no predicted samples. Use `zero_division` param
eter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Evaluation with RandomOverSampler operation

```
# Without ROS RandomOverSampler
# ===> i_ROS_op=True
sentences,labels,X,X_test_for_htest,df_htest=init_data_treatment(i_txt_proc
ess=2,i_epoch=7,i_ROS_op=True)
```

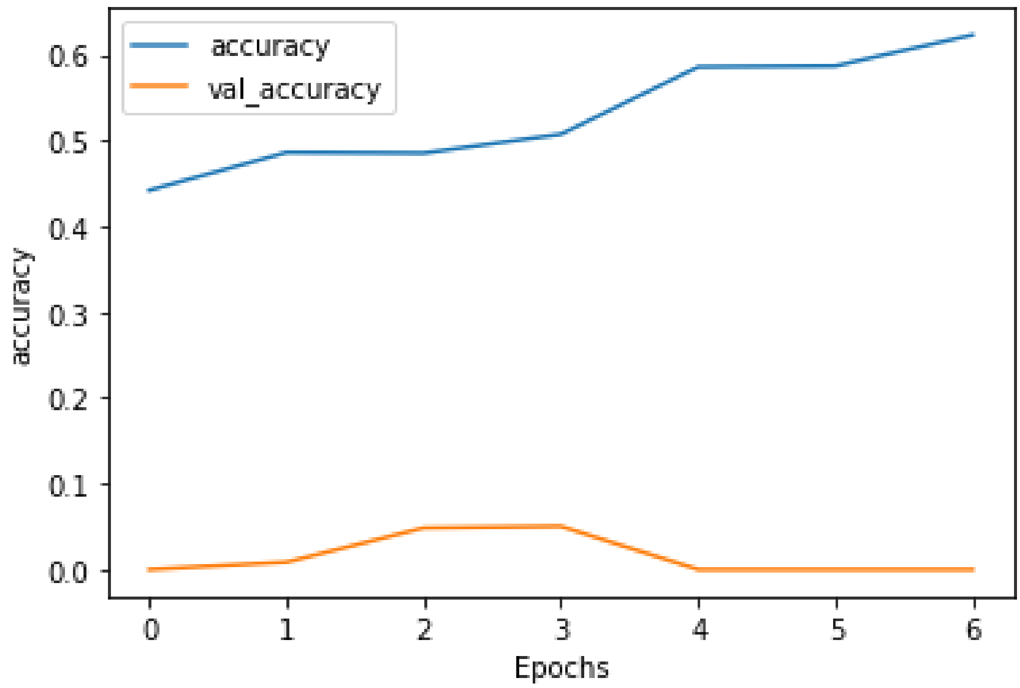
```
X.shape
```

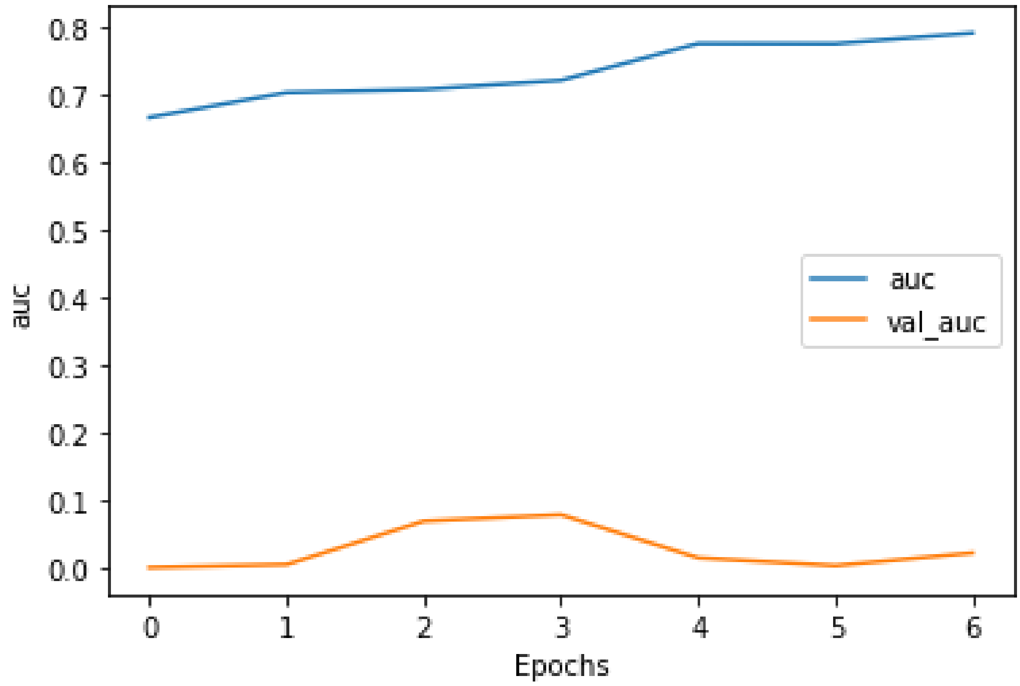
```
(11217, 1)
```

```
################################
#*** Treatment---> Case 1_layer_LSTM
###############################

#'Bi_LSTM', 1_layer_GRU, 1_layer_LSTM
choise_model='1_layer_LSTM'

# model generation
(model_tf,history_tf)=treatment_case_tensorflow(sentences,labels,choise_mod
el,tokenizer,dico_params)

# plot history graphs history for metrics accuracy and AUC
plot_graphs_history(history_tf, 'accuracy')
plot_graphs_history(history_tf, 'auc')
#plot_graphs_history(history_tf, 'loss')

# evaluation
evaluate_model_tf(model_tf,X_test_for_htest,df_htest,dico_params,tokenizer)
```

```
Model: "sequential_22"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_22 (Embedding)    (None, 120, 16)           160000

 dropout_44 (Dropout)        (None, 120, 16)           0

 lstm_10 (LSTM)              (None, 64)                20736

 dropout_45 (Dropout)        (None, 64)                0

 dense_31 (Dense)            (None, 3)                 195

 activation_22 (Activation)  (None, 3)                 0

=================================================================
Total params: 180,931
Trainable params: 180,931
Non-trainable params: 0
_________________________________________________________________
Epoch 1/7
263/263 [==============================] - 20s 67ms/step - loss: 0.5907 - a
ccuracy: 0.4421 - auc: 0.6672 - val_loss: 1.0427 - val_accuracy: 0.0000e+00
- val_auc: 0.0000e+00
Epoch 2/7
263/263 [==============================] - 17s 64ms/step - loss: 0.5645 - a
```

```
ccuracy: 0.4861 - auc: 0.7040 - val_loss: 1.0046 - val_accuracy: 0.0086 - v
al_auc: 0.0043
Epoch 3/7
263/263 [==============================] - 17s 65ms/step - loss: 0.5547 - a
ccuracy: 0.4853 - auc: 0.7088 - val_loss: 1.0283 - val_accuracy: 0.0485 - v
al_auc: 0.0687
Epoch 4/7
263/263 [==============================] - 17s 65ms/step - loss: 0.5476 - a
ccuracy: 0.5073 - auc: 0.7221 - val_loss: 1.0435 - val_accuracy: 0.0503 - v
al_auc: 0.0782
Epoch 5/7
263/263 [==============================] - 17s 64ms/step - loss: 0.5178 - a
ccuracy: 0.5858 - auc: 0.7767 - val_loss: 1.0603 - val_accuracy: 0.0000e+00
- val_auc: 0.0143
Epoch 6/7
263/263 [==============================] - 17s 65ms/step - loss: 0.5118 - a
ccuracy: 0.5869 - auc: 0.7765 - val_loss: 1.0176 - val_accuracy: 0.0000e+00
- val_auc: 0.0027
Epoch 7/7
263/263 [==============================] - 17s 65ms/step - loss: 0.4976 - a
ccuracy: 0.6230 - auc: 0.7924 - val_loss: 1.0221 - val_accuracy: 0.0000e+00
- val_auc: 0.0216
```






```
[[  7   0  17]
 [  2   0  37]
 [ 39   0 895]]
              precision    recall  f1-score   support

           0       0.15      0.29      0.19        24
           1       0.00      0.00      0.00        39
           2       0.94      0.96      0.95       934

    accuracy                           0.90       997
   macro avg       0.36      0.42      0.38       997
weighted avg       0.89      0.90      0.90       997

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1
318: UndefinedMetricWarning: Precision and F-score are ill-defined and bein
g set to 0.0 in labels with no predicted samples. Use `zero_division` param
eter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1
318: UndefinedMetricWarning: Precision and F-score are ill-defined and bein
g set to 0.0 in labels with no predicted samples. Use `zero_division` param
eter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1
318: UndefinedMetricWarning: Precision and F-score are ill-defined and bein
g set to 0.0 in labels with no predicted samples. Use `zero_division` param
eter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Bidirectional LSTM Layers

Evaluation without RandomOverSampler operation

```
# Without ROS RandomOverSampler
# ===> i_ROS_op=False
sentences,labels,X,X_test_for_htest,df_htest=init_data_treatment(i_txt_proc
ess=2,i_epoch=7,i_ROS_op=False)
```

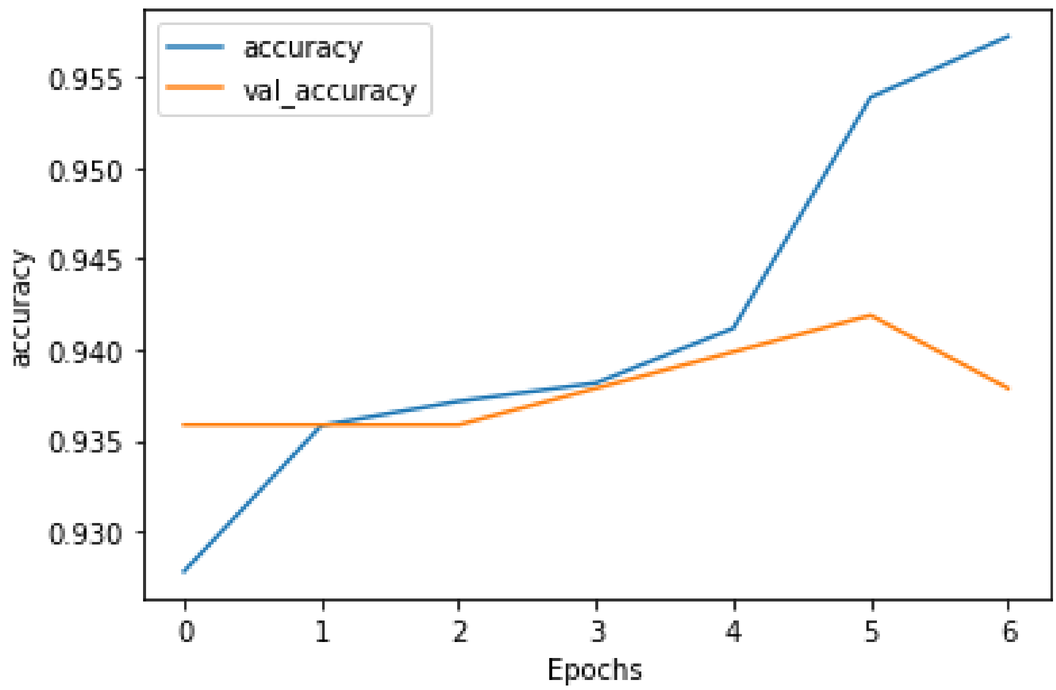
```
X.shape
```

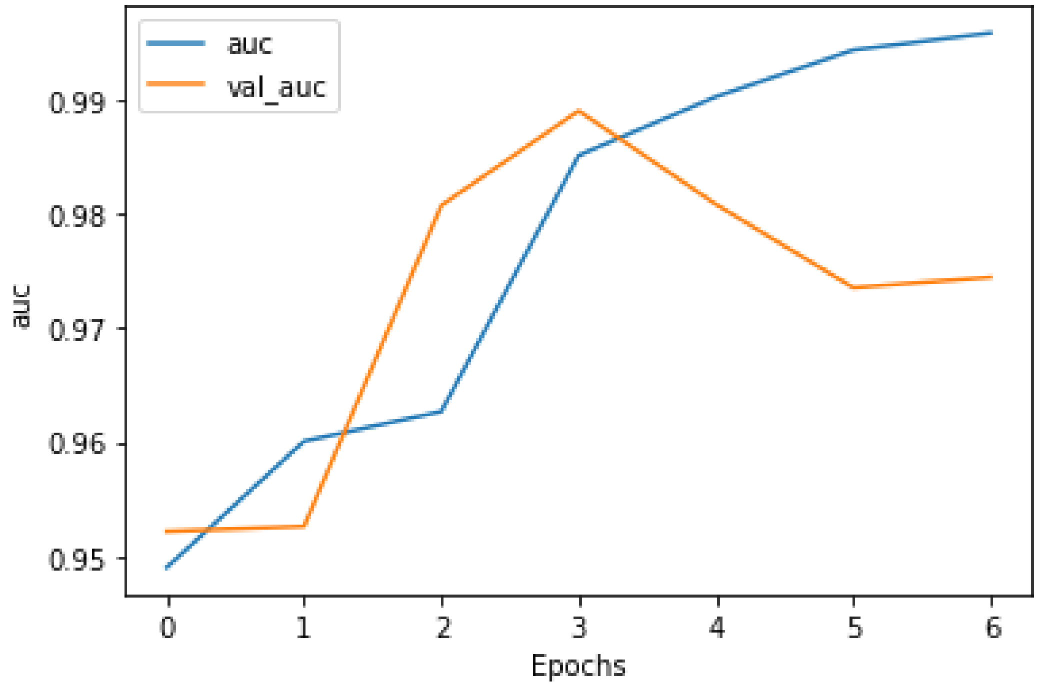
```
(3990, 1)
```

```
#################################
#*** Treatment---> Case Bi_LSTM
################################

#'Bi_LSTM', 1_layer_GRU, 1_layer_LSTM
choise_model='Bi_LSTM'

# model generation
(model_tf,history_tf)=treatment_case_tensorflow(sentences,labels,choise_mod
el,tokenizer,dico_params)

# plot history graphs history for metrics accuracy and AUC
plot_graphs_history(history_tf, 'accuracy')
plot_graphs_history(history_tf, 'auc')
#plot_graphs_history(history_tf, 'loss')

# evaluation
evaluate_model_tf(model_tf,X_test_for_htest,df_htest,dico_params,tokenizer)
```

```
Model: "sequential_23"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_23 (Embedding)    (None, 120, 16)           160000

 dropout_46 (Dropout)        (None, 120, 16)           0

 bidirectional (Bidirectiona  (None, 64)               12544
 l)

 dropout_47 (Dropout)        (None, 64)                0

 dense_32 (Dense)            (None, 24)                1560

 dropout_48 (Dropout)        (None, 24)                0

 dense_33 (Dense)            (None, 3)                 75

=================================================================
Total params: 174,179
Trainable params: 174,179
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 1/7
94/94 [==============================] - 11s 80ms/step - loss: 0.3365 - acc
uracy: 0.9278 - auc: 0.9490 - val_loss: 0.1728 - val_accuracy: 0.9359 - val
_auc: 0.9522
Epoch 2/7
94/94 [==============================] - 7s 71ms/step - loss: 0.2019 - accu
racy: 0.9358 - auc: 0.9601 - val_loss: 0.1688 - val_accuracy: 0.9359 - val_
auc: 0.9526
Epoch 3/7
94/94 [==============================] - 7s 72ms/step - loss: 0.1859 - accu
racy: 0.9372 - auc: 0.9627 - val_loss: 0.1347 - val_accuracy: 0.9359 - val_
auc: 0.9808
Epoch 4/7
94/94 [==============================] - 7s 72ms/step - loss: 0.1252 - accu
racy: 0.9382 - auc: 0.9851 - val_loss: 0.1149 - val_accuracy: 0.9379 - val_
auc: 0.9891
Epoch 5/7
94/94 [==============================] - 7s 72ms/step - loss: 0.0982 - accu
racy: 0.9412 - auc: 0.9903 - val_loss: 0.1160 - val_accuracy: 0.9399 - val_
auc: 0.9809
Epoch 6/7
94/94 [==============================] - 7s 73ms/step - loss: 0.0731 - accu
racy: 0.9539 - auc: 0.9944 - val_loss: 0.1821 - val_accuracy: 0.9419 - val_
auc: 0.9735
Epoch 7/7
94/94 [==============================] - 7s 72ms/step - loss: 0.0683 - accu
racy: 0.9572 - auc: 0.9959 - val_loss: 0.1584 - val_accuracy: 0.9379 - val_
auc: 0.9745
```






```
[[  0  12  12]
 [  0  12  27]
 [  0  10 924]]
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        24
           1       0.35      0.31      0.33        39
           2       0.96      0.99      0.97       934

    accuracy                           0.94       997
   macro avg       0.44      0.43      0.43       997
weighted avg       0.91      0.94      0.93       997

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1
318: UndefinedMetricWarning: Precision and F-score are ill-defined and bein
g set to 0.0 in labels with no predicted samples. Use `zero_division` param
eter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1
318: UndefinedMetricWarning: Precision and F-score are ill-defined and bein
g set to 0.0 in labels with no predicted samples. Use `zero_division` param
eter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1
318: UndefinedMetricWarning: Precision and F-score are ill-defined and bein
g set to 0.0 in labels with no predicted samples. Use `zero_division` param
eter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Evaluation with RandomOverSampler operation

```
# Without ROS RandomOverSampler
# ===> i_ROS_op=True
sentences,labels,X,X_test_for_htest,df_htest=init_data_treatment(i_txt_proc
ess=2,i_epoch=7,i_ROS_op=True)
```

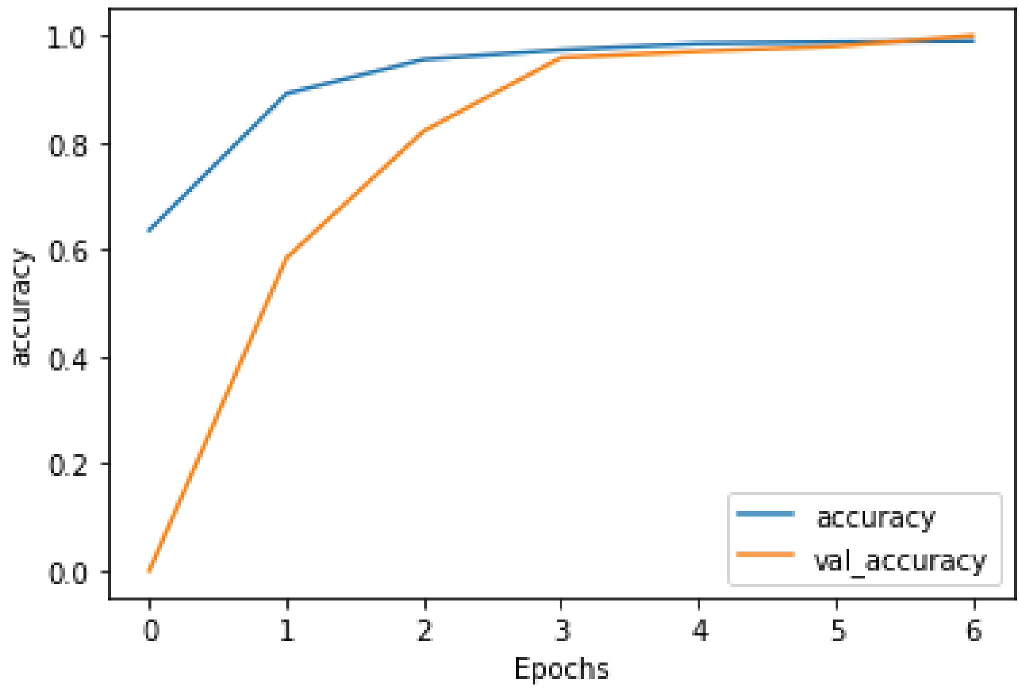
```
X.shape
```

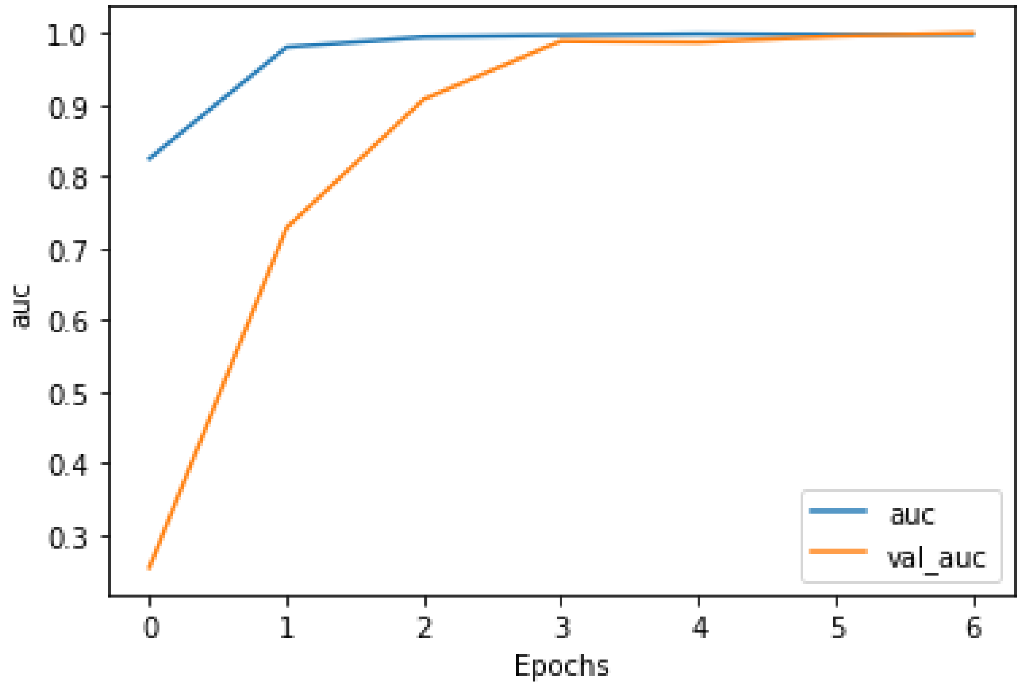
```
(11217, 1)
```

```
################################
#*** Treatment---> Case Bi_LSTM
###############################

#'Bi_LTSM', 1_layer_GRU, 1_layer_LSTM
choise_model='Bi_LSTM'

# model generation
(model_tf,history_tf)=treatment_case_tensorflow(sentences,labels,choise_mod
el,tokenizer,dico_params)

# plot history graphs history for metrics accuracy and AUC
plot_graphs_history(history_tf, 'accuracy')
plot_graphs_history(history_tf, 'auc')
#plot_graphs_history(history_tf, 'loss')

# evaluation
evaluate_model_tf(model_tf,X_test_for_htest,df_htest,dico_params,tokenizer)
```

```
Model: "sequential_24"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_24 (Embedding)    (None, 120, 16)           160000    
                                                                 
 dropout_49 (Dropout)        (None, 120, 16)           0         
                                                                 
 bidirectional_1 (Bidirectio  (None, 64)               12544     
 nal)                                                            
                                                                 
 dropout_50 (Dropout)        (None, 64)                0         
                                                                 
 dense_34 (Dense)            (None, 24)                1560      
                                                                 
 dropout_51 (Dropout)        (None, 24)                0         
                                                                 
 dense_35 (Dense)            (None, 3)                 75        
                                                                 
=================================================================
Total params: 174,179
Trainable params: 174,179
Non-trainable params: 0
_________________________________________________________________
Epoch 1/7
263/263 [==============================] - 24s 76ms/step - loss: 0.4973 - a
ccuracy: 0.6362 - auc: 0.8248 - val_loss: 1.2349 - val_accuracy: 0.0000e+00
```

```
- val_auc: 0.2545
Epoch 2/7
263/263 [==============================] - 19s 73ms/step - loss: 0.1678 - a
ccuracy: 0.8913 - auc: 0.9802 - val_loss: 0.6142 - val_accuracy: 0.5843 - v
al_auc: 0.7282
Epoch 3/7
263/263 [==============================] - 19s 72ms/step - loss: 0.0910 - a
ccuracy: 0.9554 - auc: 0.9944 - val_loss: 0.4010 - val_accuracy: 0.8214 - v
al_auc: 0.9075
Epoch 4/7
263/263 [==============================] - 19s 71ms/step - loss: 0.0596 - a
ccuracy: 0.9733 - auc: 0.9969 - val_loss: 0.1720 - val_accuracy: 0.9586 - v
al_auc: 0.9889
Epoch 5/7
263/263 [==============================] - 28s 108ms/step - loss: 0.0376 -
accuracy: 0.9848 - auc: 0.9986 - val_loss: 0.1151 - val_accuracy: 0.9704 -
val_auc: 0.9871
Epoch 6/7
263/263 [==============================] - 21s 79ms/step - loss: 0.0335 - a
ccuracy: 0.9878 - auc: 0.9977 - val_loss: 0.0708 - val_accuracy: 0.9800 - v
al_auc: 0.9955
Epoch 7/7
263/263 [==============================] - 19s 71ms/step - loss: 0.0320 - a
ccuracy: 0.9901 - auc: 0.9978 - val_loss: 0.0173 - val_accuracy: 1.0000 - v
al_auc: 1.0000
```






```
[[ 11   5   8]
 [  3  19  17]
 [  3  27 904]]
              precision    recall  f1-score   support

           0       0.65      0.46      0.54        24
           1       0.37      0.49      0.42        39
           2       0.97      0.97      0.97       934

    accuracy                           0.94       997
   macro avg       0.66      0.64      0.64       997
weighted avg       0.94      0.94      0.94       997
```

P13. Using techniques: Grid Search, Cross-Validation and Random Search

Those that have given good results are XGBoost and Bidirectional LTSM Layers. Thereafter we will optimize the parameters of the models resulting from these two approaches.

XGBoost Optimization

```
# imblearn.over_sampling.RandomOverSampler to handle imbalanced data
# ===> i_ROS_op=True
sentences,labels,X,X_test_for_htest,df_htest=init_data_treatment(i_txt_proc
ess=2,i_epoch=5,i_ROS_op=True)
```

```
# checking
len(X),len(sentences),len(labels)
```

```
(11217, 11217, 11217)
```

```
# Data preparation for fit()
y=pd.Series(labels)
if isinstance(X, pd.core.frame.DataFrame):
  X=X['reviews_title_text'] # X must be Series for train after
Xtest_h=X_test_for_htest['reviews_title_text']
ytest_h=df_htest['sentiment']
#
Xtest_h.shape,ytest_h.shape,y.shape
#
# Split() X.....
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, r
andom_state=101)
X_train.shape, X_test.shape, y_train.shape, y_test

# Tf-idf operation
from sklearn.feature_extraction.text import TfidfVectorizer
tf_idf_ROS = TfidfVectorizer()
#
# conversion of reviews in Tf-Idf score
X_train_tfidf = tf_idf_ROS.fit_transform(X_train)
X_test_tfidf=tf_idf_ROS.transform(X_test)
#print('X_test_tfidf.shape=',X_test_tfidf.shape)

from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBClassifier

#
model_xgb = XGBClassifier(n_splits=10,learning_rate =0.1, n_estimators=1000
,
                            max_depth=5,min_child_weight=1,gamma=0,subsampl
e=0.8,
                              colsample_bytree=0.8,objective= 'binary:logis
tic',
                                scale_pos_weight=1,seed=27)

tuned_parameters={ 'learning_rate' : [0.20],  # [0.05,0.10,0.15,0.20,0.25,0
.30]
                    'max_depth' : [ 6], # [ 3, 4, 5, 6, 8, 10, 12, 15]
                      'min_child_weight' : [ 1], # [ 1, 3, 5, 7 ]
                        'gamma': [ 0.1], # [ 0.0, 0.1, 0.2 , 0.3, 0.4 ]
                          'colsample_bytree' : [ 0.7 ],  # [ 0.3, 0.4, 0.5
, 0.7, 0.8,0.9]
                            'scale_pos_weight' : [1]      #[1,2,3]

                  }

#
CV_xgb=RandomizedSearchCV(cv=5, error_score='raise', estimator=model_xgb, p
aram_distributions=tuned_parameters,n_jobs=2,
                                   pre_dispatch='2*n_jobs', refit=True, ret
urn_train_score='warn',
                                       scoring=None, verbose=0)


#
```

```
CV_xgb.fit(X_train_tfidf, y_train)

CV_xgb.best_params_
#
fit_and_evaluate_model(CV_xgb,X_train_tfidf,X_test_tfidf,y_train,y_test)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_search.py:2
96: UserWarning: The total space of parameters 1 is smaller than n_iter=10.
Running 1 iterations. For exhaustive searches, use GridSearchCV.
  UserWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_search.py:2
96: UserWarning: The total space of parameters 1 is smaller than n_iter=10.
Running 1 iterations. For exhaustive searches, use GridSearchCV.
  UserWarning,

[[927   0   0]
 [  0 949   0]
 [  2  14 913]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       927
           1       0.99      1.00      0.99       949
           2       1.00      0.98      0.99       929

    accuracy                           0.99      2805
   macro avg       0.99      0.99      0.99      2805
weighted avg       0.99      0.99      0.99      2805


RandomizedSearchCV(cv=5, error_score='raise',
                   estimator=XGBClassifier(colsample_bytree=0.8, max_depth=
5,
                                           n_estimators=1000, n_splits=10,
                                           nthread=4, seed=27, subsample=0.
8),
                   n_jobs=2,
                   param_distributions={'colsample_bytree': [0.7],
                                        'gamma': [0.1], 'learning_rate': [0
.2],
                                        'max_depth': [6],
                                        'min_child_weight': [1],
                                        'nthread': [10],
                                        'scale_pos_weight': [1]},
                   return_train_score='warn')
```

```
CV_xgb.best_params_
```

```
{'colsample_bytree': 0.7,
 'gamma': 0.1,
 'learning_rate': 0.2,
 'max_depth': 6,
 'min_child_weight': 1,
 'nthread': 10,
 'scale_pos_weight': 1}
```

```
#fit_and_evaluate_model(model_xgb_ROS,X_train_tfidf,X_test_tfidf,y_train,y_
test)
evaluate_model_data_h(CV_xgb,tf_idf_ROS,Xtest_h,ytest_h)

[[ 13   3   8]
 [  2  16  21]
 [  2   8 924]]
              precision    recall  f1-score   support

           0       0.76      0.54      0.63        24
           1       0.59      0.41      0.48        39
           2       0.97      0.99      0.98       934

    accuracy                           0.96       997
   macro avg       0.78      0.65      0.70       997
weighted avg       0.95      0.96      0.95       997


 # XGBoost
'''
{'colsample_bytree': 0.7,
 'gamma': 0.2,
 'learning_rate': 0.1,
 'max_depth': 7,
 'min_child_weight': 1,
 'scale_pos_weight': 1}
'''
#
from xgboost import XGBClassifier

model_xgb_ROS = XGBClassifier(
  n_splits=10,
  learning_rate =0.1,
  n_estimators=1000,
  max_depth=7,
  min_child_weight=1,
  gamma=0.2,
  subsample=0.8,
  colsample_bytree=0.7,
  objective= 'binary:logistic',
  scale_pos_weight=1,
  seed=27)

fit_and_evaluate_model(model_xgb_ROS,X_train_tfidf,X_test_tfidf,y_train,y_t
est)
evaluate_model_data_h(model_xgb_ROS,tf_idf_ROS,Xtest_h,ytest_h)

[[927   0   0]
 [  0 949   0]
 [  3  13 913]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       927
           1       0.99      1.00      0.99       949
           2       1.00      0.98      0.99       929
```

```
    accuracy                           0.99      2805
   macro avg       0.99      0.99      0.99      2805
weighted avg       0.99      0.99      0.99      2805

[[ 14   2   8]
 [  2  15  22]
 [  1   6 927]]
              precision    recall  f1-score   support

           0       0.82      0.58      0.68        24
           1       0.65      0.38      0.48        39
           2       0.97      0.99      0.98       934

    accuracy                           0.96       997
   macro avg       0.81      0.65      0.72       997
weighted avg       0.95      0.96      0.95       997
```

```
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBClassifier

#
model_xgb = XGBClassifier(n_splits=10,learning_rate =0.1, n_estimators=1000
,
                            max_depth=5,min_child_weight=1,gamma=0,subsampl
e=0.8,
                              colsample_bytree=0.8,objective= 'binary:logis
tic',
                                scale_pos_weight=1,seed=27)

tuned_parameters={ 'learning_rate' : [0.1,0.2,0.3],  # [0.05,0.10,0.15,0.20
,0.25,0.30]
                    'max_depth' : [ 6,7], # [ 3, 4, 5, 6, 8, 10, 12, 15]
                      'min_child_weight' : [ 1], # [ 1, 3, 5, 7 ]
                        'gamma': [ 0.1,0.2], # [ 0.0, 0.1, 0.2 , 0.3, 0.4 ]
                          'colsample_bytree' : [ 0.7,0.8 ],  # [ 0.3, 0.4,
0.5 , 0.7, 0.8,0.9]
                            'scale_pos_weight' : [1]      #[1,2,3]

                  }

#
CV_xgb=RandomizedSearchCV(cv=5, error_score='raise', estimator=model_xgb, p
aram_distributions=tuned_parameters,n_jobs=2,
                                   pre_dispatch='2*n_jobs', refit=True, ret
urn_train_score='warn',
                                       scoring=None, verbose=0)

#
CV_xgb.fit(X_train_tfidf, y_train)

print('Best Params',CV_xgb.best_params_)
```

```
print("Best: %f using %s" % (CV_xgb.best_score_, CV_xgb.best_params_))
#

Best Params {'scale_pos_weight': 1, 'min_child_weight': 1, 'max_depth': 7,
'learning_rate': 0.1, 'gamma': 0.2, 'colsample_bytree': 0.7}
Best: 0.994888 using {'scale_pos_weight': 1, 'min_child_weight': 1, 'max_de
pth': 7, 'learning_rate': 0.1, 'gamma': 0.2, 'colsample_bytree': 0.7}

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-30-801e792ac1df> in <module>()
     28 print("Best: %f using %s" % (CV_xgb.best_score_, CV_xgb.best_params
_))
     29 #
---> 30 fit_and_evaluate_model(CV_xgb,X_train_tfidf,X_test_tfidf,y_train,y_
test)
     31 

<ipython-input-8-e0b730908ccc> in fit_and_evaluate_model(i_model, i_X_train
, i_X_test, i_y_train, i_y_test)
      8    #
      9    l_model=i_model
---> 10    l_model.fit(i_X_train, i_y_train)
     11    l_ypred = l_model.predict(i_X_test)
     12 

/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_search.py i
n fit(self, X, y, groups, **fit_params)
    889                 return results
    890 
--> 891             self._run_search(evaluate_candidates)
    892 
    893             # multimetric is determined here because in the case of
a callable

/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_search.py i
n _run_search(self, evaluate_candidates)
   1766         evaluate_candidates(
   1767             ParameterSampler(
-> 1768                 self.param_distributions, self.n_iter, random_state
=self.random_state
   1769             )
   1770         )

/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_search.py i
n evaluate_candidates(candidate_params, cv, more_results)
    849                     )
    850                     for (cand_idx, parameters), (split_idx, (train,
test)) in product(
--> 851                         enumerate(candidate_params), enumerate(cv.s
plit(X, y, groups))
    852                     )
    853                 )
```

```
/usr/local/lib/python3.7/dist-packages/joblib/parallel.py in __call__(self,
iterable)
   1054
   1055                 with self._backend.retrieval_context():
-> 1056                     self.retrieve()
   1057                 # Make sure that we get a last message telling us we ar
e done
   1058                 elapsed_time = time.time() - self._start_time

/usr/local/lib/python3.7/dist-packages/joblib/parallel.py in retrieve(self)
    933                 try:
    934                     if getattr(self._backend, 'supports_timeout', False
):
--> 935                         self._output.extend(job.get(timeout=self.timeou
t))
    936                     else:
    937                         self._output.extend(job.get())

/usr/local/lib/python3.7/dist-packages/joblib/_parallel_backends.py in wrap
_future_result(future, timeout)
    540             AsyncResults.get from multiprocessing."""
    541             try:
--> 542                 return future.result(timeout=timeout)
    543             except CfTimeoutError as e:
    544                 raise TimeoutError from e

/usr/lib/python3.7/concurrent/futures/_base.py in result(self, timeout)
    428                     return self.__get_result()
    429
--> 430                 self._condition.wait(timeout)
    431
    432                 if self._state in [CANCELLED, CANCELLED_AND_NOTIFIED]:

/usr/lib/python3.7/threading.py in wait(self, timeout)
    294             try:    # restore state no matter what (e.g., KeyboardInter
rupt)
    295                 if timeout is None:
--> 296                     waiter.acquire()
    297                     gotit = True
    298                 else:

KeyboardInterrupt:
```

```
CV_xgb.best_params_
```

```
{'colsample_bytree': 0.7,
 'gamma': 0.2,
 'learning_rate': 0.1,
 'max_depth': 7,
 'min_child_weight': 1,
 'scale_pos_weight': 1}
```

```
#fit_and_evaluate_model(model_xgb_ROS,X_train_tfidf,X_test_tfidf,y_train,y_
test)
evaluate_model_data_h(CV_xgb,tf_idf_ROS,Xtest_h,ytest_h)
```

```
[[ 14   2   8]
 [  2  15  22]
 [  1   6 927]]
              precision    recall  f1-score   support

           0       0.82      0.58      0.68        24
           1       0.65      0.38      0.48        39
           2       0.97      0.99      0.98       934

    accuracy                           0.96       997
   macro avg       0.81      0.65      0.72       997
weighted avg       0.95      0.96      0.95       997


#====================== Cross Validation


# k-fold cross validation evaluation of xgboost model
from numpy import loadtxt
import xgboost
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
#
# CV model
model_xgb_ROS = XGBClassifier(
  n_splits=10,
  learning_rate =0.1,
  n_estimators=1000,
  max_depth=7,
  min_child_weight=1,
  gamma=0.2,
  subsample=0.8,
  colsample_bytree=0.7,
  objective= 'binary:logistic',
  scale_pos_weight=1,
  seed=27)
kfold = KFold(n_splits=2, shuffle=True,random_state=7)
results = cross_val_score(model_xgb_ROS, X_train_tfidf, y_train, cv=kfold, 
scoring='r2')
print("Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100)
)

Accuracy: 97.74% (0.29%)

'''
>>> from sklearn import svm, cross_validation, datasets
>>> iris = datasets.load_iris()
>>> X, y = iris.data, iris.target
>>> model = svm.SVC()
>>> cross_validation.cross_val_score(model, X, y, scoring='wrong_choice')
Traceback (most recent call last):
ValueError: 'wrong_choice' is not a valid scoring value. Valid options are 
['accuracy', 'adjusted_rand_score', 'average_precision', 'f1', 'log_loss', 
'mean_absolute_error', 'mean_squared_error', 'precision', 'r2', 'recall', '
```

```
roc_auc']
'''
```

```
results
```

```
array([0.9976247 , 0.99524941, 0.99762188, 0.99524376, 0.99643282,
       0.99762188, 0.99167658, 0.99524376, 0.9940547 , 0.99643282])
```

Bidirectional LTSM Layers Optimization

GridSearchCV

```
sentences,labels,X,X_test,df_htest=init_data_treatment(i_txt_process=1,i_ep
och=7,i_ROS_op=True)
```

```
X.shape
```

```
(11217, 1)
```

Searching the Best Optimizer by GridSearchCV

```
import numpy
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier

#model_tf.compile(loss='binary_crossentropy', optimizer = Adam_def, metrics
=['accuracy','AUC'])

# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)

# Function to create model, required for KerasClassifier
def create_model(optimizer='adam'):

  # create model
  l_model_tf = Sequential()
  l_model_tf=get_model_tf(choise_model,dico_params)

  # Compile model
  l_model_tf.compile(loss='binary_crossentropy', optimizer=optimizer, metri
cs=['accuracy','AUC'])

  return l_model_tf


# create model
model_tf = KerasClassifier(build_fn=create_model, epochs=2, batch_size=10,
verbose=0)

# define the grid search parameters
optimizer = ['Adagrad', 'Adadelta', 'Adam', 'Nadam'] # ['SGD', 'RMSprop', '
Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam']
```

```
# Def grid param
param_grid = dict(optimizer=optimizer)

grid = GridSearchCV(estimator=model_tf, param_grid=param_grid, n_jobs=-1, c
v=3)
grid_result = grid.fit(training_padded, training_labels)

# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_para
ms_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: Deprecatio
nWarning: KerasClassifier is deprecated, use Sci-Keras (https://github.com/
adriangb/scikeras) instead. See https://www.adriangb.com/scikeras/stable/mi
gration.html for help migrating.

Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_6 (Embedding)     (None, 120, 16)           160000

 dropout_18 (Dropout)        (None, 120, 16)           0

 bidirectional_6 (Bidirectio  (None, 64)               12544
 nal)

 dropout_19 (Dropout)        (None, 64)                0

 dense_14 (Dense)            (None, 24)                1560

 dropout_20 (Dropout)        (None, 24)                0

 dense_15 (Dense)            (None, 3)                 75

=================================================================
Total params: 174,179
Trainable params: 174,179
Non-trainable params: 0
_________________________________________________________________
```

Best: 0.888493 using {'optimizer': 'Nadam'}

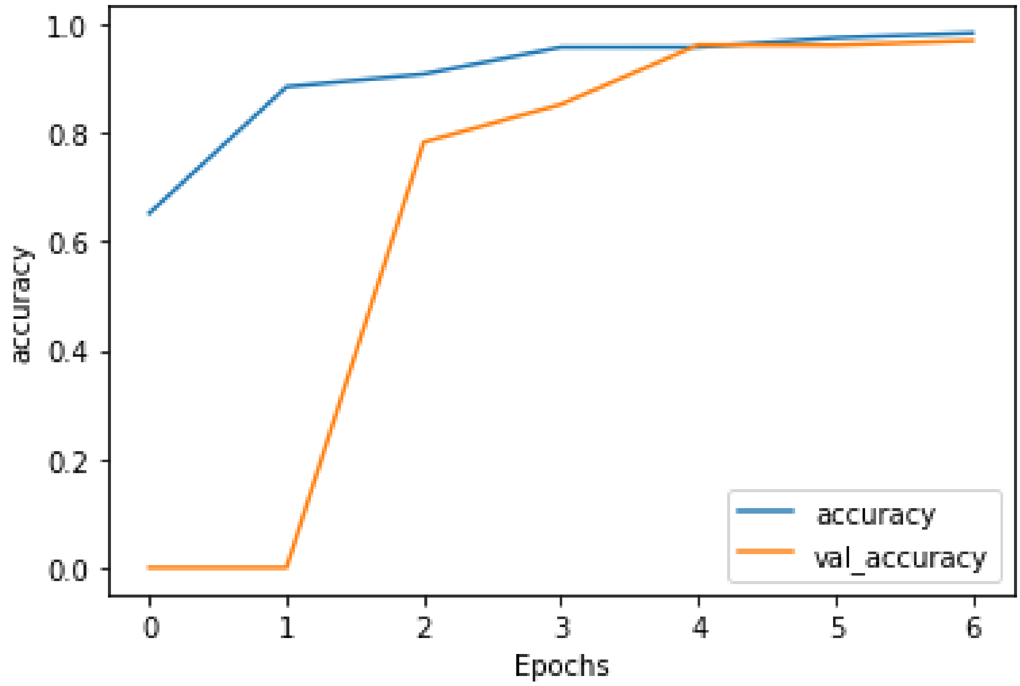
```
0.140989 (0.182509) with: {'optimizer': 'Adagrad'}
0.066334 (0.053025) with: {'optimizer': 'Adadelta'}
0.880052 (0.111663) with: {'optimizer': 'Adam'}
0.888493 (0.117543) with: {'optimizer': 'Nadam'}
```

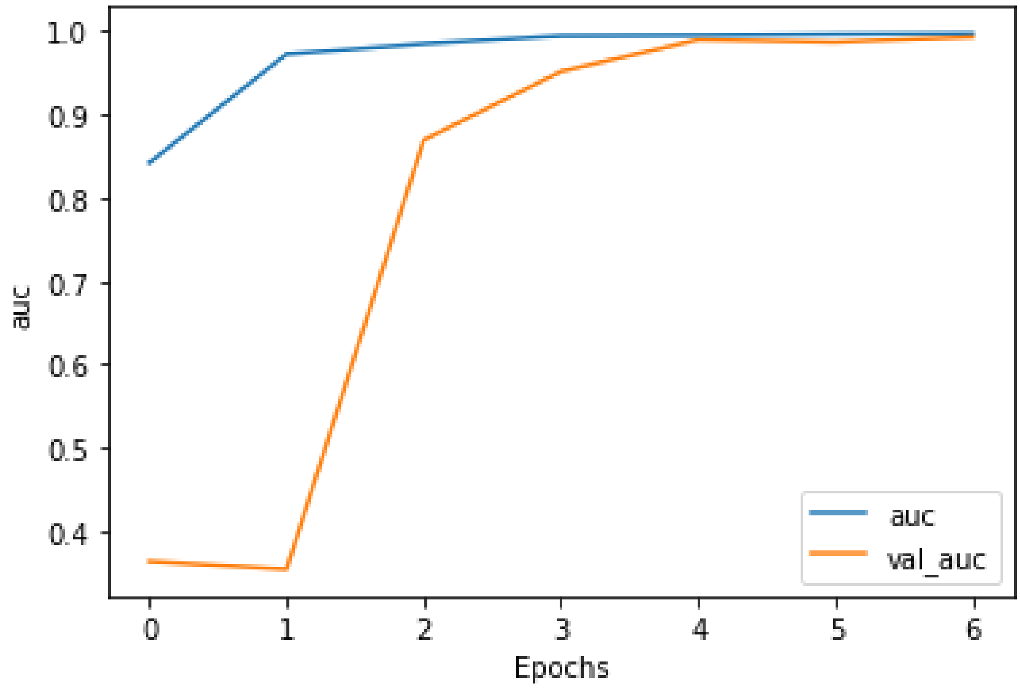
Test with optimizer = 'Nadam'

```
#==================== Test avec optimizer = 'Nadam'

################################
#*** Treatment---> Case Bi_LSTM
###############################

#'Bi_LSTM', 1_layer_GRU, 1_layer_LSTM
choise_model='Bi_LSTM'

# model generation
#(model_tf,history_tf)=treatment_case_tensorflow3(sentences,labels,choise_m
odel,tokenizer,dico_params)
'''
Adam_def=tf.keras.optimizers.Adam(
  learning_rate=0.01,
  beta_1=0.8,
  beta_2=0.8,
  epsilon=1e-07,
  amsgrad=False,
  name='Adam1'
)

SGD_def=tf.keras.optimizers.SGD(
  learning_rate=0.03,
  momentum=0.06,
  nesterov=False,
  name='SGD1'
)
'''
# Train the model --> Appel de treatment_case_tensorflow_without_fit()
training_padded, training_labels, testing_padded, testing_labels = treatmen
t_case_tensorflow_without_fit(sentences,labels,choise_model,tokenizer,dico_
params)
model_tf=get_model_tf(choise_model,dico_params)

# model_tf.compile(loss='binary_crossentropy', optimizer = Adam_def, metric
s=['accuracy','AUC'])
model_tf.compile(loss='binary_crossentropy', optimizer = 'Nadam', metrics=[
'accuracy','AUC'])

history_tf = model_tf.fit(training_padded, training_labels, epochs=dico_par
ams.get('NUM_EPOCHS'), validation_data=(testing_padded, testing_labels))

# plot history graphs history for metrics accuracy and AUC
plot_graphs_history(history_tf, 'accuracy')
plot_graphs_history(history_tf, 'auc')
#plot_graphs_history(history_tf, 'loss')

# evaluation
evaluate_model_tf(model_tf,X_test_for_htest,df_htest,dico_params,tokenizer)
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

```
=================================================================
 embedding_2 (Embedding)     (None, 120, 16)           160000

 dropout_6 (Dropout)         (None, 120, 16)           0

 bidirectional_2 (Bidirectio  (None, 64)               12544
 nal)

 dropout_7 (Dropout)         (None, 64)                0

 dense_4 (Dense)             (None, 24)                1560

 dropout_8 (Dropout)         (None, 24)                0

 dense_5 (Dense)             (None, 3)                 75

=================================================================
Total params: 174,179
Trainable params: 174,179
Non-trainable params: 0
_________________________________________________________________
Epoch 1/7
263/263 [==============================] - 35s 114ms/step - loss: 0.4687 -
accuracy: 0.6519 - auc: 0.8421 - val_loss: 1.0990 - val_accuracy: 0.0000e+0
0 - val_auc: 0.3644
Epoch 2/7
263/263 [==============================] - 23s 86ms/step - loss: 0.1996 - a
ccuracy: 0.8843 - auc: 0.9723 - val_loss: 0.8469 - val_accuracy: 0.0000e+00
- val_auc: 0.3551
Epoch 3/7
263/263 [==============================] - 22s 85ms/step - loss: 0.1487 - a
ccuracy: 0.9079 - auc: 0.9846 - val_loss: 0.4706 - val_accuracy: 0.7825 - v
al_auc: 0.8694
Epoch 4/7
263/263 [==============================] - 22s 85ms/step - loss: 0.0839 - a
ccuracy: 0.9572 - auc: 0.9942 - val_loss: 0.3292 - val_accuracy: 0.8520 - v
al_auc: 0.9515
Epoch 5/7
263/263 [==============================] - 22s 85ms/step - loss: 0.0784 - a
ccuracy: 0.9573 - auc: 0.9948 - val_loss: 0.1328 - val_accuracy: 0.9615 - v
al_auc: 0.9895
Epoch 6/7
263/263 [==============================] - 22s 85ms/step - loss: 0.0513 - a
ccuracy: 0.9744 - auc: 0.9963 - val_loss: 0.1060 - val_accuracy: 0.9611 - v
al_auc: 0.9866
Epoch 7/7
263/263 [==============================] - 22s 86ms/step - loss: 0.0386 - a
ccuracy: 0.9835 - auc: 0.9970 - val_loss: 0.0712 - val_accuracy: 0.9693 - v
al_auc: 0.9927
```






```
[[ 10   4  10]
 [  2  16  21]
 [  3  17 914]]
              precision    recall  f1-score   support

           0       0.67      0.42      0.51        24
           1       0.43      0.41      0.42        39
           2       0.97      0.98      0.97       934

    accuracy                           0.94       997
   macro avg       0.69      0.60      0.64       997
```

```
weighted avg       0.94      0.94      0.94       997
```

GridSearchCV

```
sentences,labels,X,X_test,df_htest=init_data_treatment(i_txt_process=2,i_ep
och=7,i_ROS_op=True)

#####################################
#####################################
"""# Optimization"""

#**********************************
#*** Treatment Tensorflow cas optimization
#**********************************
#
from tensorflow.keras.preprocessing.sequence import pad_sequences
#

#'Bi_LSTM', 1_layer_GRU, 1_layer_LSTM
choise_model='Bi_LSTM'

# model generation
(X_tf,y_tf,X_test_ft,y_test_tf) = treatment_case_tensorflow_without_fit(sen
tences,labels,choise_model,tokenizer,dico_params)


import numpy
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
#from keras.optimizers import SGD

# Function to create model, required for KerasClassifier
def create_model(learn_rate=0.02, beta_1=0.9, beta_2=0.999):
  # create model
  l_model = Sequential()
  # l_model = get_def_model_tf(choise_model,dico_params)
  l_model = get_def_model_tf('Bi_LSTM', dico_params)


  # Compile model
  #l_optimizer = tf.keras.optimizers.SGD(learning_rate=learn_rate, momentum
=momentum)
  l_optimizer = tf.keras.optimizers.Nadam(learning_rate=0.001, beta_1=0.9,
beta_2=0.999)
  #
  l_model.compile(loss='binary_crossentropy', optimizer=l_optimizer, metric
s=['accuracy','AUC'])  # accuracy

  return l_model

# fix random seed for reproducibility
```

```
seed = 195
numpy.random.seed(seed)


# create model
model = KerasClassifier(build_fn=create_model, epochs=7, batch_size=10, ver
bose=0)
# define the grid search parameters
learn_rate = [0.01,0.05,0.02,0.03]    #[0.001, 0.01, 0.1, 0.2, 0.3]
beta_1 =  [0.6,0.7,0.8,0.9]      #[0.0, 0.2, 0.4,0.5, 0.6, 0.7,0.8, 0.9]
beta_2 =  [0.8, 0.9,0.999]
#
param_grid = dict(learn_rate=learn_rate, beta_1=beta_1, beta_2=beta_2)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3
)
grid_result = grid.fit(X_tf, y_tf)

# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_para
ms_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))

sentences,labels,X,X_test,df_htest=init_data_treatment(i_txt_process=1,i_ep
och=7,i_ROS_op=True)
```

Run-on another machine

Best: 0.934617 using {'beta_1': 0.8, 'beta_2': 0.9, 'learn_rate': 0.01}

0.912149 (0.047117) with: {'beta_1': 0.6, 'beta_2': 0.8, 'learn_rate': 0.005}
0.912625 (0.066235) with: {'beta_1': 0.6, 'beta_2': 0.8, 'learn_rate': 0.01}
0.917974 (0.061969) with: {'beta_1': 0.6, 'beta_2': 0.8, 'learn_rate': 0.02}
0.894912 (0.066826) with: {'beta_1': 0.6, 'beta_2': 0.8, 'learn_rate': 0.03}
0.883738 (0.108573) with: {'beta_1': 0.6, 'beta_2': 0.9, 'learn_rate': 0.005}
0.887542 (0.112371) with: {'beta_1': 0.6, 'beta_2': 0.9, 'learn_rate': 0.01}
0.881954 (0.102344) with: {'beta_1': 0.6, 'beta_2': 0.9, 'learn_rate': 0.02}
0.893367 (0.094631) with: {'beta_1': 0.6, 'beta_2': 0.9, 'learn_rate': 0.03}
0.912625 (0.077778) with: {'beta_1': 0.6, 'beta_2': 0.999, 'learn_rate': 0.005}
0.879339 (0.085904) with: {'beta_1': 0.6, 'beta_2': 0.999, 'learn_rate': 0.01}
0.888255 (0.070357) with: {'beta_1': 0.6, 'beta_2': 0.999, 'learn_rate': 0.02}
0.894080 (0.076643) with: {'beta_1': 0.6, 'beta_2': 0.999, 'learn_rate': 0.03}
0.891108 (0.071428) with: {'beta_1': 0.7, 'beta_2': 0.8, 'learn_rate': 0.005}
0.877437 (0.084572) with: {'beta_1': 0.7, 'beta_2': 0.8, 'learn_rate': 0.01}
0.891227 (0.095036) with: {'beta_1': 0.7, 'beta_2': 0.8, 'learn_rate': 0.02}
0.896576 (0.096311) with: {'beta_1': 0.7, 'beta_2': 0.8, 'learn_rate': 0.03}
0.873157 (0.110406) with: {'beta_1': 0.7, 'beta_2': 0.9, 'learn_rate': 0.005}
0.885758 (0.111050) with: {'beta_1': 0.7, 'beta_2': 0.9, 'learn_rate': 0.01}
0.881598 (0.097570) with: {'beta_1': 0.7, 'beta_2': 0.9, 'learn_rate': 0.02}
0.900024 (0.097466) with: {'beta_1': 0.7, 'beta_2': 0.9, 'learn_rate': 0.03}
0.889444 (0.087827) with: {'beta_1': 0.7, 'beta_2': 0.999, 'learn_rate': 0.005}
0.930456 (0.034301) with: {'beta_1': 0.7, 'beta_2': 0.999, 'learn_rate': 0.01}
0.881360 (0.102462) with: {'beta_1': 0.7, 'beta_2': 0.999, 'learn_rate': 0.02}
0.871731 (0.108791) with: {'beta_1': 0.7, 'beta_2': 0.999, 'learn_rate': 0.03}

0.881835 (0.114078) with: {'beta_1': 0.8, 'beta_2': 0.8, 'learn_rate': 0.005}
0.885640 (0.110072) with: {'beta_1': 0.8, 'beta_2': 0.8, 'learn_rate': 0.01}
0.904066 (0.077740) with: {'beta_1': 0.8, 'beta_2': 0.8, 'learn_rate': 0.02}
0.874108 (0.111169) with: {'beta_1': 0.8, 'beta_2': 0.8, 'learn_rate': 0.03}
0.891108 (0.095983) with: {'beta_1': 0.8, 'beta_2': 0.9, 'learn_rate': 0.005}
0.934617 (0.041224) with: {'beta_1': 0.8, 'beta_2': 0.9, 'learn_rate': 0.01}
0.888374 (0.097020) with: {'beta_1': 0.8, 'beta_2': 0.9, 'learn_rate': 0.02}
0.876010 (0.067078) with: {'beta_1': 0.8, 'beta_2': 0.9, 'learn_rate': 0.03}
0.881954 (0.113692) with: {'beta_1': 0.8, 'beta_2': 0.999, 'learn_rate': 0.005}
0.886472 (0.090537) with: {'beta_1': 0.8, 'beta_2': 0.999, 'learn_rate': 0.01}
0.878982 (0.100332) with: {'beta_1': 0.8, 'beta_2': 0.999, 'learn_rate': 0.02}
0.909772 (0.074948) with: {'beta_1': 0.8, 'beta_2': 0.999, 'learn_rate': 0.03}
0.873514 (0.105463) with: {'beta_1': 0.9, 'beta_2': 0.8, 'learn_rate': 0.005}
0.876367 (0.094260) with: {'beta_1': 0.9, 'beta_2': 0.8, 'learn_rate': 0.01}
0.876129 (0.105932) with: {'beta_1': 0.9, 'beta_2': 0.8, 'learn_rate': 0.02}
0.897171 (0.069340) with: {'beta_1': 0.9, 'beta_2': 0.8, 'learn_rate': 0.03}
0.885521 (0.058794) with: {'beta_1': 0.9, 'beta_2': 0.9, 'learn_rate': 0.005}
0.888849 (0.112226) with: {'beta_1': 0.9, 'beta_2': 0.9, 'learn_rate': 0.01}
0.895388 (0.080439) with: {'beta_1': 0.9, 'beta_2': 0.9, 'learn_rate': 0.02}
0.886472 (0.116346) with: {'beta_1': 0.9, 'beta_2': 0.9, 'learn_rate': 0.03}
0.873871 (0.082706) with: {'beta_1': 0.9, 'beta_2': 0.999, 'learn_rate': 0.005}
0.908226 (0.079621) with: {'beta_1': 0.9, 'beta_2': 0.999, 'learn_rate': 0.01}
0.910604 (0.076966) with: {'beta_1': 0.9, 'beta_2': 0.999, 'learn_rate': 0.02}
0.874108 (0.096740) with: {'beta_1': 0.9, 'beta_2': 0.999, 'learn_rate': 0.03}

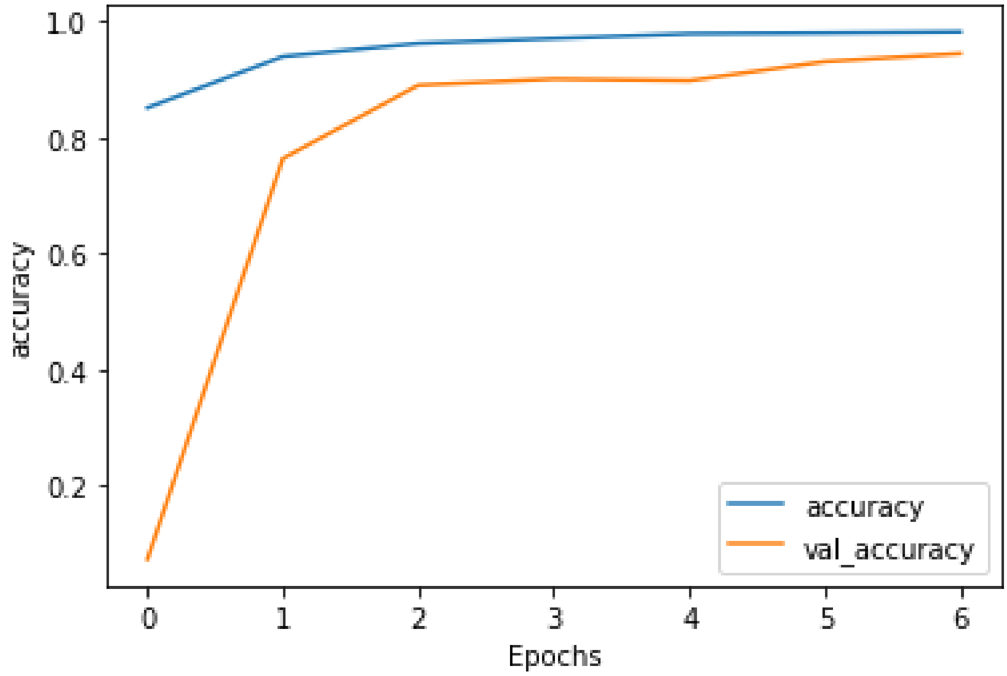
Process finished with exit code 0

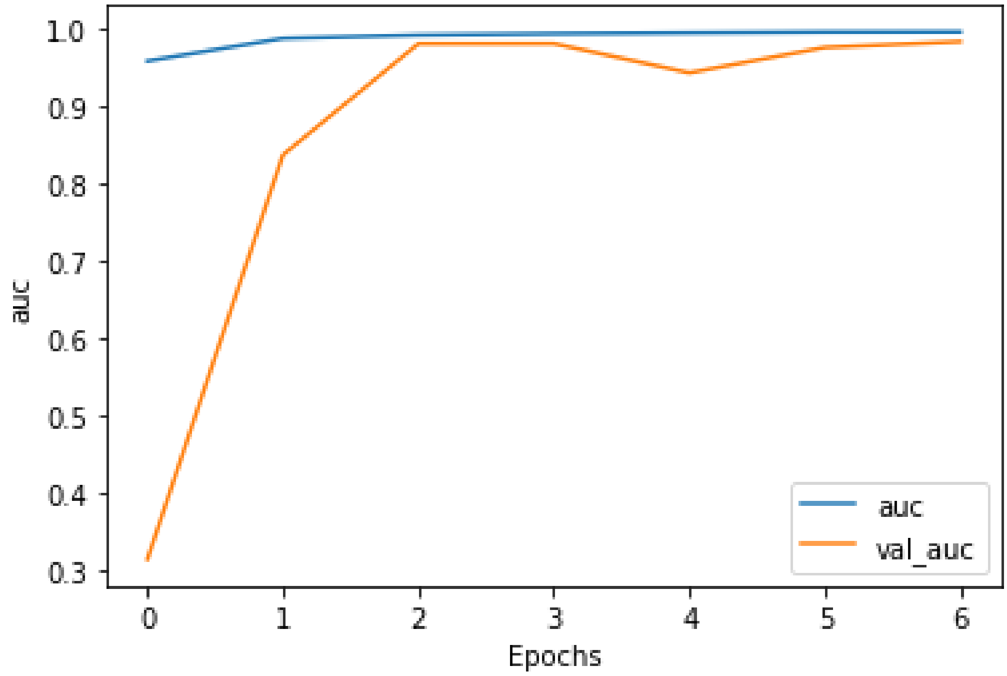
```
#========================== Test avec optimizer = 'Nadam'


################################
#*** Treatment---> Case Bi_LSTM

#'Bi_LSTM', 1_layer_GRU, 1_layer_LSTM
choise_model='Bi_LSTM'

# Train the model --> Appel de treatment_case_tensorflow_without_fit()
training_padded, training_labels, testing_padded, testing_labels = treatmen
t_case_tensorflow_without_fit(sentences,labels,choise_model,tokenizer,dico_
params)
model_tf=get_model_tf(choise_model,dico_params)
#

#*********
optimizer_Nadam = tf.keras.optimizers.Nadam(learning_rate=0.01, beta_1=0.8,
beta_2=0.9)
#*********

model_tf.compile(loss='binary_crossentropy', optimizer = optimizer_Nadam, m
etrics=['accuracy','AUC'])

history_tf = model_tf.fit(training_padded, training_labels, epochs=dico_par
ams.get('NUM_EPOCHS'), validation_data=(testing_padded, testing_labels))

# plot history graphs history for metrics accuracy and AUC
plot_graphs_history(history_tf, 'accuracy')
```

```
plot_graphs_history(history_tf, 'auc')
#plot_graphs_history(history_tf, 'loss')

# evaluation
evaluate_model_tf(model_tf,X_test_for_htest,df_htest,dico_params,tokenizer)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 120, 16)           160000

 dropout (Dropout)           (None, 120, 16)           0

 bidirectional (Bidirectiona  (None, 64)               12544
 l)

 dropout_1 (Dropout)         (None, 64)                0

 dense (Dense)               (None, 24)                1560

 dropout_2 (Dropout)         (None, 24)                0

 dense_1 (Dense)             (None, 3)                 75

=================================================================
Total params: 174,179
Trainable params: 174,179
Non-trainable params: 0
_________________________________________________________________
Epoch 1/7
263/263 [==============================] - 31s 98ms/step - loss: 0.2395
- accuracy: 0.8513 - auc: 0.9579 - val_loss: 2.2275 - val_accuracy:
0.0724 - val_auc: 0.3157
Epoch 2/7
263/263 [==============================] - 26s 98ms/step - loss: 0.1144
- accuracy: 0.9396 - auc: 0.9872 - val_loss: 0.6465 - val_accuracy:
0.7636 - val_auc: 0.8369
Epoch 3/7
263/263 [==============================] - 25s 94ms/step - loss: 0.0781
- accuracy: 0.9623 - auc: 0.9914 - val_loss: 0.1503 - val_accuracy:
0.8902 - val_auc: 0.9803
Epoch 4/7
263/263 [==============================] - 25s 94ms/step - loss: 0.0640
- accuracy: 0.9708 - auc: 0.9931 - val_loss: 0.1684 - val_accuracy:
0.9009 - val_auc: 0.9804
Epoch 5/7
263/263 [==============================] - 25s 94ms/step - loss: 0.0547
- accuracy: 0.9788 - auc: 0.9943 - val_loss: 0.3150 - val_accuracy:
0.8980 - val_auc: 0.9427
Epoch 6/7
263/263 [==============================] - 26s 100ms/step - loss:
0.0524 - accuracy: 0.9803 - auc: 0.9953 - val_loss: 0.1879 -
val_accuracy: 0.9308 - val_auc: 0.9756
Epoch 7/7

263/263 [==============================] - 25s 95ms/step - loss: 0.0504
- accuracy: 0.9822 - auc: 0.9956 - val_loss: 0.1234 - val_accuracy:
```

```
0.9447 - val_auc: 0.9829
```






```
[[  9   9   6]
 [  1  17  21]
 [  1  15 918]]
              precision    recall  f1-score   support

           0       0.82      0.38      0.51        24
           1       0.41      0.44      0.43        39
           2       0.97      0.98      0.98       934

    accuracy                           0.95       997
   macro avg       0.73      0.60      0.64       997
```

```
weighted avg       0.95      0.95      0.94       997
```

V- Topic Modeling:

```
import scipy as sp;
import sklearn;
import sys;
from nltk.corpus import stopwords;
import nltk;
from gensim.models import ldamodel
import gensim.corpora;
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransform
er;
from sklearn.decomposition import NMF;
from sklearn.preprocessing import normalize;
import pickle;

all_reviews  = df["reviews.text"].astype('str').tolist()
all_reviews = [text_processing2(cleanText(doc)).split() for doc in all_revi
ews]

#number of topics we will cluster for: num_topics=10 and num_topn=15
num_topics = 8
num_topn=12
```

P14. Identification of similar clusters by: Latent Dirchlette Allocation LDA scikit-learn technique

```
# --- Case LDA

# list of topn words by category
# generate dataframe
def get_lda_topics(model, num_topics):
    word_dict = {};
    for i in range(num_topics):
        words = model.show_topic(i, topn = num_topn);
        word_dict['Topic # ' + '{:02d}'.format(i+1)] = [i[0] for i in words
];
    return pd.DataFrame(word_dict);
#----------------------

id2word = gensim.corpora.Dictionary(all_reviews);

corpus = [id2word.doc2bow(text) for text in all_reviews];

lda = ldamodel.LdaModel(corpus=corpus, id2word=id2word, num_topics=num_topi
```

```
cs);

# generating topics
df_t=get_lda_topics(lda, num_topics)
df_t
```

	Topic # 01	Topic # 02	Topic # 03	Topic # 04	Topic # 05	Topic # 06	Topic # 07	Topic # 08
0	love	tablet	echo	great	love	love	kindl	use
1	great	great	tablet	amazon	one	bought	tablet	great
2	use	good	love	music	use	tablet	love	love
3	echo	use	great	use	purchas	like	great	easi
4	alexa	easi	use	play	bought	use	one	product
5	amazon	love	get	video	got	set	fire	read
6	show	price	devic	alexa	tablet	nice	old	bought
7	like	kid	amazon	product	new	play	bought	alexa
8	tablet	screen	need	app	realli	good	use	light
9	one	would	buy	love	enjoy	time	read	work
10	kindl	recommend	plus	echo	gift	year	year	book
11	thing	read	want	tablet	kindl	product	purchas	play

P15. Identification of similar clusters by: Non-Negative Matrix Factorization NMF scikit-learn technique

```
# --- Case NMF

# list of topn words by category
# generate dataframe
def get_nmf_topics(model, n_top_words):

    # the word ids obtained need to be reverse-mapped to the words
    # so we can print the topic names.
    feat_names = vectorizer.get_feature_names()

    word_dict = {};
    for i in range(num_topics):

        # for each topic, obtain the largest values,
        # and add the words they map to into the dictionary.
        words_ids = model.components_[i].argsort()[:-num_topn - 1:-1]
        words = [feat_names[key] for key in words_ids]
        word_dict['Topic # ' + '{:02d}'.format(i+1)] = words;

    return pd.DataFrame(word_dict);
#-------------------------

train_headlines_sentences = [' '.join(text) for text in all_reviews]
```

```
vectorizer = CountVectorizer(analyzer='word', max_features=5000);
x_counts = vectorizer.fit_transform(train_headlines_sentences);

transformer = TfidfTransformer(smooth_idf=False);
x_tfidf = transformer.fit_transform(x_counts);

xtfidf_norm = normalize(x_tfidf, norm='l1', axis=1)

model = NMF(n_components=num_topics, init='nndsvd');

# fit the model
model.fit(xtfidf_norm)

# generating topics
df_t=get_nmf_topics(model, num_topn)
df_t
```

	Topic # 01	Topic # 02	Topic # 03	Topic # 04	Topic # 05	Topic # 06	Topic # 07	Topic # 08
0	great	love	easi	tablet	echo	good	old	kindl
1	work	bought	use	kid	alexa	product	year	read
2	price	gift	set	price	show	recommend	grandson	book
3	product	daughter	product	need	music	would	bought	fire
4	kid	son	fun	app	home	price	perfect	game
5	gift	absolut	setup	perfect	like	friend	purchas	like
6	sound	christma	super	game	one	buy	one	replac
7	valu	got	learn	nice	plus	qualiti	christma	play
8	well	granddaught	navig	amazon	amazon	high	enjoy	size
9	camera	kid	problem	daughter	light	excel	son	one
10	addit	wife	item	littl	screen	definit	yr	better
11	deal	grandson	simpl	play	smart	time	happi	purchas

FUNCTIONS & INITIALIZATION

```
def reinit_input_df():
  l_df=df_init.copy(deep=True)
  l_df_htest=df_htest_init.copy(deep=True)
  l_df_test=df_test_init.copy(deep=True)
  return l_df,l_df_htest,l_df_test

def fc_mapping(i_x):
  r_value=0
  if i_x=='Positive':
    r_value=2
  else:
```

```
    if i_x=='Neutral':
      r_value=1
    else:
      if i_x=='Negative':
        r_value=0
      else:
        r_value=i_x
  #
  return r_value

def retraite_df(i_df, i_level_review=0, i_fusion_reviews_title_text=True):
  #
  l_df=i_df
  l_sentences=[]
  # rename(): reviews.title ------> reviews_title
  l_df.rename(columns={'reviews.title': 'reviews_title'}, inplace=True)
  l_df.rename(columns={'reviews.text': 'reviews_text'}, inplace=True)
  # drop rows
  l_df.dropna()
  l_indexNames = l_df[ (l_df['reviews_title'].isna()) | (l_df['reviews_text
'].isna()) ].index    #
  l_df.drop(l_indexNames , inplace=True)
  #
  # mapping values of sentiment colomn
  if 'sentiment' in l_df.columns:
    l_df['sentiment'] = l_df['sentiment'].apply(fc_mapping)
  #
  if i_fusion_reviews_title_text:
    l_sentences=np.array(l_df.reviews_title.astype(str))+'. '+ np.array(l_d
f.reviews_text.astype(str))
    if i_level_review==1:
      l_sentences=np.array(l_df.reviews_text.astype(str))
    #
    if i_level_review==2:
      l_sentences=np.array(l_df.reviews_title.astype(str))
    #
    l_sentences=list(l_sentences)
    l_df['reviews_title_text']=l_sentences
    l_sentences=[]
  return l_df
```

P10. Definition of a score evaluation function based on the sentiment of sentences. It will be the evaluation tool to see the improvement of the models and to compare them.

```
#========================#

##########################################
### Evaluation function No tensorflow model
##########################################
# Used to Evaluate models
```

```
from sklearn.metrics import confusion_matrix, classification_report
#
def fit_and_evaluate_model(i_model,i_X_train,i_X_test,i_y_train,i_y_test):
  #
  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



##################################
### Evaluation tensorflow model
##################################
from sklearn.metrics import confusion_matrix, classification_report
#
def evaluate_model_tf(i_model,i_df_for_sequences,i_df_for_labels, i_dico_pa
rams, i_tokenizer):
  Xpad,ylab=get_params_input_toPredict(i_df_for_sequences,i_df_for_labels,i
_dico_params, i_tokenizer)
  ylab_pred=i_model.predict(Xpad)
  ylab_pred_lst=[np.argmax(ylab_pred[i,:]) for i in range(len(ylab_pred))]
  print(confusion_matrix(ylab, ylab_pred_lst))
  print(classification_report(ylab, ylab_pred_lst))

#=========================#



# Evaluate Hidden Data
# i_transformer(i_Xtest_h) in order to_predict()
def evaluate_model_data_h(i_model,i_transformer,i_Xtest_h,i_ytest_h):
  #
  l_model=i_model
  l_Xtest_h=i_transformer.transform(i_Xtest_h)
  l_ypred = l_model.predict(l_Xtest_h)

  print(confusion_matrix(i_ytest_h, l_ypred))
  print(classification_report(i_ytest_h, l_ypred))

  return None


# Plot history graph
def plot_graphs_history(i_history, i_metric_str):
  plt.plot(i_history.history[i_metric_str])
  plt.plot(i_history.history['val_'+i_metric_str])
  plt.xlabel("Epochs")
  plt.ylabel(i_metric_str)
  plt.legend([i_metric_str, 'val_'+i_metric_str])
```

```
  plt.show()
  return None

def get_params_input_toPredict(i_df_for_sequences,i_df_for_labels, i_dico_p
arams, i_tokenizer):
  #
  l_sentences=list(i_df_for_sequences.reviews_title_text)
  l_labels=list(i_df_for_labels.sentiment)
  #
  # Generate and pad the training sequences
  l_sentences = i_tokenizer.texts_to_sequences(l_sentences)
  l_seq_padded = pad_sequences(l_sentences, maxlen=i_dico_params.get('max_l
ength'), padding=i_dico_params.get('padding_type'), truncating=i_dico_param
s.get('trunc_type'))

  # Convert the labels lists into numpy arrays
  l_labels = np.array(l_labels)
  np_utils.to_categorical(l_labels, i_dico_params.get('nb_classes'))

  #
  return l_seq_padded,l_labels    # sequences ou sentences

import numpy as np
import tensorflow as tf
from keras.layers.core import Dense, Dropout, Activation

def get_model_tf(i_key_model,i_dico_params):
  #
  tf.random.set_seed=195
  # Parameters
  nb_classes=i_dico_params.get('nb_classes')
  embedding_dim = i_dico_params.get('embedding_dim')
  lstm_dim = i_dico_params.get('lstm_dim')
  dense_dim = i_dico_params.get('dense_dim')
  vocab_size = i_dico_params.get('vocab_size')
  max_length = i_dico_params.get('max_length')

  # Model Definition with LSTM
  if i_key_model=='Bi_LSTM':
    l_model = tf.keras.Sequential([
        tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=m
ax_length),
        tf.keras.layers.Dropout(rate=0.4, seed=195),
        tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(lstm_dim)),
        tf.keras.layers.Dropout(rate=0.6, seed=195),
        tf.keras.layers.Dense(dense_dim, activation='relu'),
        #tf.keras.layers.Dense(1, activation='sigmoid')
        tf.keras.layers.Dropout(rate=0.4),
        tf.keras.layers.Dense(nb_classes, activation='softmax')
    ])
  #
  if i_key_model=='1_layer_LSTM':
    l_model = tf.keras.Sequential()
    l_model.add(tf.keras.layers.Embedding(vocab_size, embedding_dim, input_
```

```
length=max_length))
    l_model.add(tf.keras.layers.Dropout(rate=0.4, seed=195))
    l_model.add(tf.keras.layers.LSTM(2*lstm_dim))
    l_model.add(tf.keras.layers.Dropout(rate=0.6, seed=195))
    l_model.add(tf.keras.layers.Dense(nb_classes))
    l_model.add(Activation('softmax'))
  #
  if i_key_model=='1_layer_GRU':
    l_model = tf.keras.Sequential()
    l_model.add(tf.keras.layers.Embedding(vocab_size, embedding_dim, input_
length=max_length))
    l_model.add(tf.keras.layers.Dropout(rate=0.4, seed=195))
    l_model.add(tf.keras.layers.GRU(120))
    l_model.add(tf.keras.layers.Dropout(rate=0.6, seed=195))
    l_model.add(tf.keras.layers.Dense(nb_classes))
    l_model.add(Activation('softmax'))
    #
  if i_key_model=='1_layer_Dense':
    l_model = tf.keras.Sequential()
    l_model.add(tf.keras.layers.Embedding(vocab_size, embedding_dim, input_
length=max_length))
    l_model.add(tf.keras.layers.Dropout(rate=0.4, seed=195))
    l_model.add(tf.keras.layers.Dense(10*nb_classes))
    l_model.add(tf.keras.layers.Dropout(rate=0.6, seed=195))
    l_model.add( tf.keras.layers.Flatten())
    l_model.add(tf.keras.layers.Dense(nb_classes))
    l_model.add(Activation('softmax'))

  # Set the training parameters
  l_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['a
ccuracy','AUC'])
  # tf.keras.metrics.AUC(),metrics=['accuracy','AUC']

  # Print the model summary
  l_model.summary()
  #
  return l_model

#*********************************
#*** Treatment Tensorflow
#*********************************
#
from tensorflow.keras.preprocessing.sequence import pad_sequences
#
def treatment_case_tensorflow(i_sentences,i_labels,i_key_model,i_tokenizer,
i_dico_params):

  tf.random.set_seed=195
  #--------------------
  #-----Split operation
  #--------------------
  ratio_training=75/100
  training_size=int(len(i_sentences)*ratio_training)
```

```
  # Split the sentences
  training_sentences = i_sentences[0:training_size]
  testing_sentences = i_sentences[training_size:]

  # Split the labels
  training_labels = i_labels[0:training_size]
  testing_labels = i_labels[training_size:]

  #---------------------
  #-----Padding operation
  #---------------------
  #

  # Generate the word index dictionary
  i_tokenizer.fit_on_texts(training_sentences)
  word_index = tokenizer.word_index

  # Generate and pad the training sequences
  training_sequences = i_tokenizer.texts_to_sequences(training_sentences)
  training_padded = pad_sequences(training_sequences, maxlen=i_dico_params.
get('max_length'), padding=i_dico_params.get('padding_type'),
                                                                          tru
ncating=i_dico_params.get('trunc_type'))

  # Generate and pad the testing sequences
  testing_sequences = i_tokenizer.texts_to_sequences(testing_sentences)
  testing_padded = pad_sequences(testing_sequences, maxlen=i_dico_params.ge
t('max_length'), padding=i_dico_params.get('padding_type'),

truncating=i_dico_params.get('trunc_type'))

  # Convert the labels lists into numpy arrays
  training_labels = np.array(training_labels)
  testing_labels = np.array(testing_labels)

  # One-Hot Encoding of y_train and y_test
  from keras.utils import np_utils
  nb_classes=3
  #
  training_padded1=training_padded
  testing_padded1=training_padded
  #
  training_labels1=training_labels
  testing_labels1=testing_labels
  #
  training_labels = np_utils.to_categorical(training_labels, nb_classes)
  testing_labels = np_utils.to_categorical(testing_labels, nb_classes)


  # Train the model
  l_model=get_model_tf(i_key_model,i_dico_params)
  l_history = l_model.fit(training_padded, training_labels, epochs=i_dico_p
arams.get('NUM_EPOCHS'), validation_data=(testing_padded, testing_labels))
```

```
  return (l_model,l_history)

import string
from nltk.corpus import stopwords
#
import nltk
nltk.download('stopwords')
#

def text_processing(i_text):

    #Takes in a string of text, then performs the following:
    #1. Remove all punctuation 2. Remove all stopwords
    #3. Return the cleaned text as a list of words

    l_txt=i_text
    l_txt = [char for char in i_text if char not in string.punctuation]
    l_txt = ''.join(l_txt)
    #[word.lower() for word in l_txt.split() if word.lower() not in stopwor
ds.words('english')]
    #l_txt = ''.join(l_txt)

    return l_txt
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
import string
from nltk.corpus import stopwords
#
import nltk
nltk.download('stopwords')
#

def text_processing2(i_text):

    l_txt=i_text
    l_txt = [char for char in l_txt if char not in string.punctuation]
    l_txt = ''.join(l_txt)

    l_txt = l_txt.split()
    l_txt =[word.lower() for word in l_txt]
    l_txt =' '.join(l_txt)

    #StopwordRemoval
    #from nltk.corpus import stopwords
    #l_txt = [word for word in l_txt if word.lower() not in stopwords.words
('english')]
    #l_txt = ''.join(l_txt)

    return l_txt
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
from nltk.tokenize import RegexpTokenizer
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.corpus import wordnet
import nltk
#
nltk.download('wordnet')
nltk.download('stopwords')

def text_processing3(i_text):
    l_text = i_text.lower() # Convert to Lowercase
    l_words = l_text.split() # Tokenize
    l_words = [w for w in l_words if not w in stopwords.words('english')] #
Removing stopwords

    # Lemmatizing
    for pos in [wordnet.NOUN, wordnet.VERB, wordnet.ADJ, wordnet.ADV]:
        l_words = [WordNetLemmatizer.lemmatize(x, pos) for x in l_words]
    return " ".join(l_words)
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
from bs4 import BeautifulSoup
from nltk.stem import SnowballStemmer, WordNetLemmatizer
import re
def text_processing4(raw_text, remove_stopwords=True, stemming=True, split_
text=False ):

    text = BeautifulSoup(raw_text, 'lxml').get_text()  # remove htm
    letters_only = re.sub("[^a-zA-Z]", " ", text)  # remove non-char
    words = letters_only.lower().split() #  to lower

    # remove stopword
    if remove_stopwords:
        stops = set(stopwords.words("english"))
        words = [w for w in words if not w in stops]

    if stemming==True: # stemming
        # stemmer = PorterStemmer()
        stemmer = SnowballStemmer('english')
        words = [stemmer.stem(w) for w in words]

    if split_text==True:
        return (words)

    return( " ".join(words))

# RandomOverSampler to handle imbalanced data
from imblearn.over_sampling import RandomOverSampler
```

```
#
def ROS_operation(i_X,i_Y):
  ros = RandomOverSampler(random_state=0)
  l_X,l_Y=ros.fit_resample(i_X,i_Y)
  print("X_result.shape,Y_result.shape", l_X.shape , l_Y.shape)
  return l_X, l_Y

########################################
### init_data(i_txt_process,i_ROS_op=True)
# i_txt_process=1 ==> text_processing
# i_txt_process=2 ==> text_processing2
########################################
def init_data_treatment(i_txt_process=1,i_epoch=7,i_ROS_op=True):

  # Initial data
  df,df_htest,df_test=reinit_input_df()
  print('df.shape,df_htest.shape,df_test.shape')
  print(df.shape,df_htest.shape,df_test.shape,'\n')

  #i_fusion_reviews_title_text=True
  #
  #level_review=
  #                0:reviews_text+'. '+reviews_title
  #                1: reviews_text
  #                2: reviews_title
  level_review=0
  #
  df=retraite_df(df,i_level_review=level_review)
  df_htest=retraite_df(df_htest,i_level_review=level_review,i_fusion_review
s_title_text=False)  # reviews_title_text not necessary
  df_test=retraite_df(df_test,i_level_review=level_review)
  #
  df.info(),df_htest.info(),df_test.info()

  # X, X_test, Y
  X=pd.DataFrame(df['reviews_title_text'])
  X_test=pd.DataFrame(df_test['reviews_title_text'])
  #
  # treatment text
  if i_txt_process==1:
    X=pd.DataFrame(df['reviews_title_text'].apply(text_processing))
    X_test=pd.DataFrame(df_test['reviews_title_text'].apply(text_processing
))
  #
  if i_txt_process==2:
    X=pd.DataFrame(df['reviews_title_text'].apply(text_processing2))
    X_test=pd.DataFrame(df_test['reviews_title_text'].apply(text_processing
2))
  #
  Y=df[['sentiment']]

  # RandomOverSampler to handle imbalanced data
  if i_ROS_op:
    X,Y=ROS_operation(X,Y)
```

```
    print('\nX.shape,Y.shape')
    print(X.shape,Y.shape)

  sentences=list(X.reviews_title_text)
  labels=list(Y.sentiment)
  print('\nlabels example....', 'sentences example....')
  print (labels[:10],'', sentences[4:5])

  # epochs number
  dico_params['NUM_EPOCHS']=i_epoch

  # Initialize the Tokenizer class
  print('\nInitialyzing tokenizer.... ')
  tokenizer = Tokenizer(num_words=dico_params.get('vocab_size'), oov_token=
dico_params.get('oov_tok'))

  print('\n')
  X.info(),X_test.info()
  return sentences,labels, X, X_test,df_htest

# --------- For Optimization model tf
def treatment_case_tensorflow_without_fit(i_sentences,i_labels,i_key_model,
i_tokenizer,i_dico_params):

  tf.random.set_seed=195
  #---------------------
  #-----Split operation
  #---------------------
  ratio_training=75/100
  training_size=int(len(i_sentences)*ratio_training)

  # Split the sentences
  training_sentences = i_sentences[0:training_size]
  testing_sentences = i_sentences[training_size:]

  # Split the labels
  training_labels = i_labels[0:training_size]
  testing_labels = i_labels[training_size:]

  #---------------------
  #-----Padding operation
  #---------------------
  #

  # Generate the word index dictionary
  i_tokenizer.fit_on_texts(training_sentences)
  word_index = tokenizer.word_index

  # Generate and pad the training sequences
  training_sequences = i_tokenizer.texts_to_sequences(training_sentences)
  training_padded = pad_sequences(training_sequences, maxlen=i_dico_params.
get('max_length'), padding=i_dico_params.get('padding_type'),
                                                                         tru
ncating=i_dico_params.get('trunc_type'))
```

```
  # Generate and pad the testing sequences
  testing_sequences = i_tokenizer.texts_to_sequences(testing_sentences)
  testing_padded = pad_sequences(testing_sequences, maxlen=i_dico_params.ge
t('max_length'), padding=i_dico_params.get('padding_type'),

truncating=i_dico_params.get('trunc_type'))

  # Convert the labels lists into numpy arrays
  training_labels = np.array(training_labels)
  testing_labels = np.array(testing_labels)

  # One-Hot Encoding of y_train and y_test
  from keras.utils import np_utils
  nb_classes=3
  #
  training_padded1=training_padded
  testing_padded1=training_padded
  #
  training_labels1=training_labels
  testing_labels1=testing_labels
  #
  training_labels = np_utils.to_categorical(training_labels, nb_classes)
  testing_labels = np_utils.to_categorical(testing_labels, nb_classes)

  return training_padded, training_labels, testing_padded, testing_labels

import numpy as np
import tensorflow as tf
#from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.layers.core import Dense, Dropout, Activation, Lambda
from keras.utils import np_utils
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
#
from tensorflow.keras.preprocessing.text import Tokenizer
import nltk
nltk.download('stopwords')
from sklearn.feature_extraction.text import CountVectorizer
#
#TF-IDF
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorize
r
from xgboost import XGBClassifier


# Global varibales
X=df
X_test=df_test
#
Y=df[['sentiment']]
#
```

```
# sentences, labels
sentences=[]
labels=[]

dico_params={
      #l_dico=dict()
      'nb_classes':3,
      'lstm_dim':32,
      'embedding_dim':16,
      'dense_dim':24,
      #
      'vocab_size':10000,
      'max_length':120,
      'trunc_type':'post',
      'padding_type':'post',
      'oov_tok': "<OOV>",
      'NUM_EPOCHS':7
        }
def get_dico_params():
  return dico_params
#
# Initialize the Tokenizer class
tokenizer = Tokenizer(num_words=dico_params.get('vocab_size'), oov_token=di
co_params.get('oov_tok'))
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
#======== Final Step Initialization =============#

# Without ROS RandomOverSampler
# ===> i_ROS_op=False
sentences,labels,X,X_test_for_htest,df_htest=init_data_treatment(i_txt_proc
ess=2,i_epoch=12,i_ROS_op=False)

# Without ROS RandomOverSampler
# ===> i_ROS_op=True
sentences,labels,X,X_test_for_htest,df_htest=init_data_treatment(i_txt_proc
ess=2,i_epoch=12,i_ROS_op=True)

# Checking
X.shape
```

```
(3990, 1)
```

```
# Data preparation for fit()
y=pd.Series(labels)
if isinstance(X, pd.core.frame.DataFrame):
  X=X['reviews_title_text'] # X must be Series for train after
Xtest_h=X_test_for_htest['reviews_title_text']
ytest_h=df_htest['sentiment']
#
Xtest_h.shape,ytest_h.shape,y.shape
#
```

```
# Split() X.....
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, r
andom_state=101)
X_train.shape, X_test.shape, y_train.shape, y_test

# Tf-idf operation
from sklearn.feature_extraction.text import TfidfVectorizer
tf_idf_ROS = TfidfVectorizer()
#
# conversion of reviews in Tf-Idf score
X_train_tfidf = tf_idf_ROS.fit_transform(X_train)
X_test_tfidf=tf_idf_ROS.transform(X_test)
#print('X_test_tfidf.shape=',X_test_tfidf.shape)

#================================================================
```