MACHINE LEARNING – LAB 3

Outputs

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s12	s13	s14	s15	s16	s17	s18	s19	s20	s21
0	1	1.0	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	...	521.66	2388.02	8138.62	8.4195	0.03	392	2388	100.0	39.06	23.4190
1	1	2.0	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	...	522.28	2388.07	8131.49	8.4318	0.03	392	2388	100.0	39.00	23.4236
2	1	3.0	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	...	522.42	2388.03	8133.23	8.4178	0.03	390	2388	100.0	38.95	23.3442
3	1	4.0	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	14.62	...	522.86	2388.08	8133.83	8.3682	0.03	392	2388	100.0	38.88	23.3739
4	1	5.0	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	14.62	...	522.19	2388.04	8133.80	8.4294	0.03	393	2388	100.0	38.90	23.4044

5 rows × 26 columns

Figure 1 - train

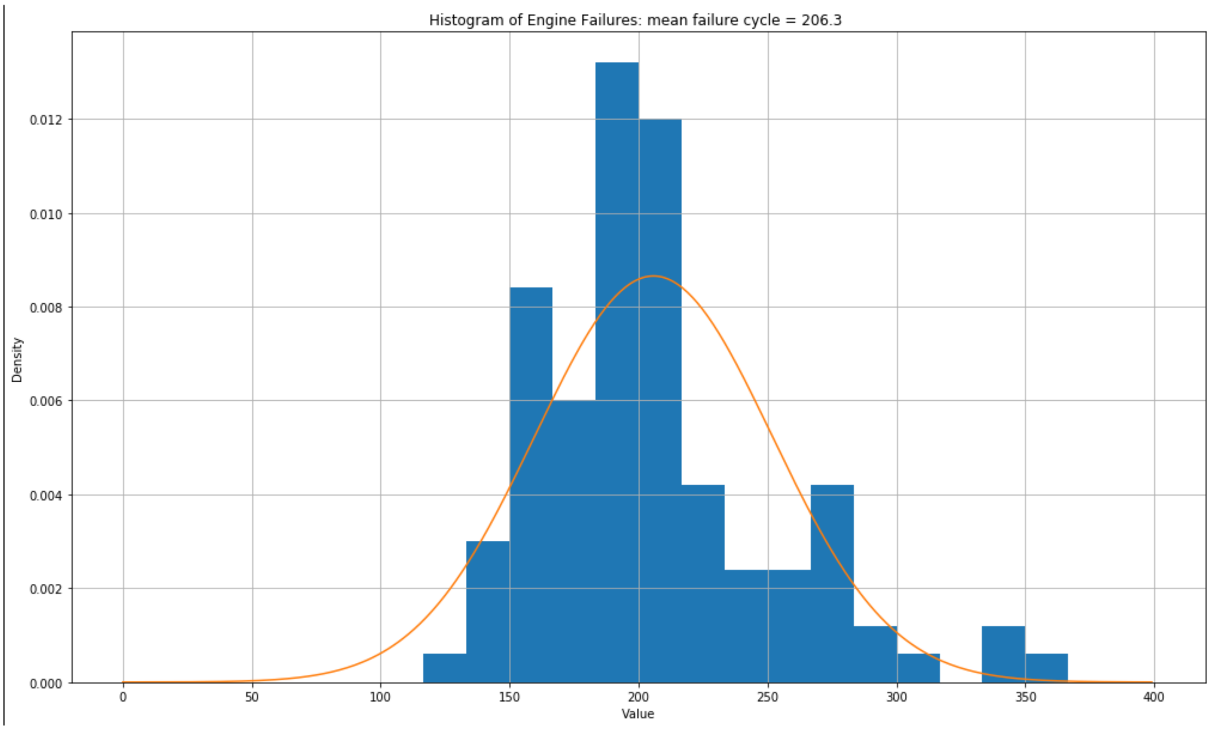

Figure 2 - Histogram of machine failures

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s14	s15	s16	s17	s18	s19	s20	s21	RUL
0	1	1.0	0.459770	0.166667	0.0	0.0	0.183735	0.406802	0.309757	0.0	...	0.199608	0.363986	0.0	0.333333	0.0	0.0	0.713178	0.724662	303.0
1	1	2.0	0.609195	0.250000	0.0	0.0	0.283133	0.453019	0.352633	0.0	...	0.162813	0.411312	0.0	0.333333	0.0	0.0	0.666667	0.731014	302.0
2	1	3.0	0.252874	0.750000	0.0	0.0	0.343373	0.369523	0.370527	0.0	...	0.171793	0.357445	0.0	0.166667	0.0	0.0	0.627907	0.621375	301.0
3	1	4.0	0.540230	0.500000	0.0	0.0	0.343373	0.256159	0.331195	0.0	...	0.174889	0.166603	0.0	0.333333	0.0	0.0	0.573643	0.662386	300.0
4	1	5.0	0.390805	0.333333	0.0	0.0	0.349398	0.257467	0.404625	0.0	...	0.174734	0.402078	0.0	0.416667	0.0	0.0	0.589147	0.704502	299.0
5	1	6.0	0.252874	0.416667	0.0	0.0	0.268072	0.292784	0.272113	0.0	...	0.169832	0.330512	0.0	0.250000	0.0	0.0	0.651163	0.652720	298.0

Figure 3 - test

```
Confusion Matrix for LR:
 [[18530  1959]
 [    0   142]]

Accuracy: 0.9050458048567689

Precision: 0.06758686339838173

Recall: 1.0
```

Figure 4 - Confusion matrix

When we run the python file, we can predict the number of days before a machine fails by choosing its number. For instance, if we put 80, the algorithm will return show us that the machine will fail after 189 days.

```
input a engine-number for prediction : 80
     Cycle  LogisticR
144    154          0
145    155          0
146    156          0
147    157          0
148    158          0
149    159          1
150    160          1
151    161          1
152    162          1
153    163          1
154    164          1
155    165          1
156    166          1
157    167          1
158    168          1
159    169          1
160    170          1
161    171          1
162    172          1
163    173          1
164    174          1
165    175          1
166    176          1
167    177          1
168    178          1
169    179          1
170    180          1
171    181          1
172    182          1
173    183          1
It will fail at 189 days.
```

Figure 5 - Final outputs : the prediction of days before a failure

Code source

```
import pandas as pd
import numpy as np
import os
import keras
import keras.backend as K
from keras.layers.core import Activation
from keras.models import Sequential, load_model
from keras.layers import Dense, Dropout, LSTM
from keras.wrappers.scikit_learn import KerasClassifier
import scipy
from scipy.stats import norm
import boto3

# get rid of deprecated warnings
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

import sklearn
from sklearn import preprocessing
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, precision_score, recall_score,
confusion_matrix
from sklearn import linear_model


import matplotlib as mlab
import matplotlib.pyplot as plt
mlab.rcParams['figure.figsize']=(17,10)

# get the training file and call its handler "train"
train = pd.read_csv('train_FD001.txt',sep=" " ,header = None)
test = pd.read_csv('test_FD001.txt',sep =" ",header = None)

if train.empty:
    raise Exception('No data found!')

# remove some columns and add titles
train.drop(train.columns[[26,27]],axis=1,inplace=True)
operational_columns = ['setting1','setting2','setting3']
observational_columns =
['s1','s2','s3','s4','s5','s6','s7','s8','s9','s10','s11','s12','s13','s14','s15'
,'s16','s17','s18','s19','s20','s21']

train.columns = ['id','cycle'] + operational_columns + observational_columns
test = train
train.head()

# draw the histogram with the average cycle of failure "mu" using:
# norm.fit and acipy.stats.norm.pdf
# plt.hist

Data1= train.groupby(['id'], sort=False,as_index=False)['cycle'].max()
max_scatter = list(Data1['cycle'])
l_1 = np.linspace(0, 400, 25)
l_2 = np.linspace(0,400,400)

mu,std=scipy.stats.norm.fit(max_scatter)
plt.hist(max_scatter,bins=l_1,normed='true')

pdf = scipy.stats.norm.pdf
```

```
plt.plot(pdf(l_2, loc=mu, scale=std))

plt.xlabel('Value')
plt.ylabel('Density')
plt.title('Histogram of Engine Failures: mean failure cycle = %.1f ' %(mu))
plt.grid(True)
plt.show()


# prepare the dataset

LOOKBACK_LENGTH = 10 # number of cycles in the past to analyse on a rolling basis
DAYS_IN_ADVANCE = 30 # number of cycles we consider before the engine fail

# get the "truth" data file to be used as the test dataset and call it "truth"

truth =
pd.read_csv('D:\\Nelly\\Documents\\ISEP\\A3\\MachineLearning\\RUL_FD001.txt',sep=
" " ,header = None)
truth.drop(truth.columns[[1]],axis=1,inplace=True)

# for a given engine, RUL = cycle at failure - current cycle
# we add this parameter as a column to the left of the training data table
# then we drop the max column that becomes useless
train.head()
rul = pd.DataFrame(train.groupby('id')['cycle'].max()).reset_index()
rul.columns = ['id','max']
train = train.merge(rul, on=['id'], how='left')

train['RUL'] = train['max'] - train['cycle']
train.drop('max', axis=1, inplace=True)

# normalize the data in settings and sensors columns

train['cycle_norm'] = train['cycle']
cols_normalize = train.columns.difference(['id','cycle','RUL'])
min_max_scaler = preprocessing.MinMaxScaler()
norm_train_df =
pd.DataFrame(min_max_scaler.fit_transform(train[cols_normalize]),columns=cols_nor
malize,index=train.index)
join_df = train[train.columns.difference(cols_normalize)].join(norm_train_df)
train = join_df.reindex(columns = train.columns)

train.head(40)

# generate column max for test data

rul = pd.DataFrame(test.groupby('id')['cycle'].max()).reset_index()
rul.columns = ['id','max']
truth.columns = ['more']
truth['id'] = truth.index + 1
truth['max'] = rul['max'] + truth['more']
truth.drop('more',axis=1,inplace=True)

truth.head()

# generate test['RUL'] for test data using max and cycle

test = test.merge(truth, on=['id'], how='left')
test['RUL'] = test['max'] - test['cycle']
test.drop('max', axis=1, inplace=True)

# normalize test data with MinMax normalization as above

test['cycle_norm'] = test['cycle']
```

```
cols_normalize = test.columns.difference(['id','cycle','RUL'])
min_max_scaler = preprocessing.MinMaxScaler()
norm_test_df =
pd.DataFrame(min_max_scaler.fit_transform(test[cols_normalize]),columns=cols_norm
alize,index=test.index)
join_df = test[test.columns.difference(cols_normalize)].join(norm_test_df)
test = join_df.reindex(columns = test.columns)

test.head(40)

# we want deux classes: 0 or 1 (no need for maintenance or maintenance needed)
train['Y'] = np.where(train['RUL'] <= DAYS_IN_ADVANCE, 1, 0)
test['Y'] = np.where(test['RUL'] <= DAYS_IN_ADVANCE, 1, 0)

feature_columns = operational_columns + ['cycle_norm'] + observational_columns
train.to_csv(r'train.csv')
test.to_csv(r'test.csv')
# train and test the model
# First method: Logistic regression

#train_Y=ans
train_Y = train['Y']
#train_rolling = train.groupby('id').apply(pd.DataFrame.rolling,LOOKBACK_LENGTH,
min_periods=1)
train_rolling = train.groupby('id')
train_rolling = train_rolling.rolling(window=LOOKBACK_LENGTH,
min_periods=1).mean()
train_rolling.to_csv(r'train_rolling.csv')
#print(train_rolling['Y'].tail(40))

train_rolling = train_rolling.drop('cycle', axis=1)
#train_rolling['Y'] = train_Y

# Y is the variable to predict according to X
train_rolling.to_csv(r'rolling.csv')
X = train_rolling.drop(['Y','RUL'], axis=1)
Y = train_rolling['Y'].fillna(0)
Y = Y.astype('int')
print(Y)
Y.to_csv(r'fitting.csv')

# create and "fit" or train the model with the training data using
linear_model.LogisticRegression
# call the model "lr_model"

from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import f_regression
from sklearn import preprocessing, linear_model

lr_model =  linear_model.LogisticRegression()
lr_model.fit(X, Y)

# print Coef of fitting
print(lr_model.coef_)

# print intercept
print(lr_model.intercept_ )
predicted_classes = lr_model.predict(X)
np.savetxt("predict_train.csv", predicted_classes, delimiter=",")
accuracy = accuracy_score(Y,predicted_classes)
prec = precision_score(Y, predicted_classes)
recall = recall_score(Y, predicted_classes)
cm_1 = confusion_matrix(Y, predicted_classes)
print(cm_1)
print('accuracy, prec and recall = ',accuracy,prec,recall)
```

```
# prepare test data for prediction
test_y = test['Y']
#test or train?
all_test = test.groupby('id')
all_test = all_test.rolling(window=LOOKBACK_LENGTH, min_periods=1).mean()
all_test = all_test.drop('cycle', axis=1)
#all_test['Y'] = test_y

# define features to predict

X_test_lr = all_test.drop(['Y','RUL'], axis=1)
Y_test_lr = all_test['Y'].fillna(0)
Y_test_lr = Y_test_lr.astype('int')

# run the model for prediction
predictions = lr_model.predict(X_test_lr)

# return model evaluation metrics using accuracy_score

logistic_acc = accuracy_score(Y_test_lr, predictions)
logistic_prec = precision_score(Y_test_lr, predictions)
logistic_recall = recall_score(Y_test_lr, predictions)

cm = confusion_matrix(Y_test_lr,predictions)
print('\nConfusion Matrix for LR:\n', cm)
print('\nAccuracy: {}'.format(logistic_acc))
print('\nPrecision: {}'.format(logistic_prec))
print('\nRecall: {}'.format(logistic_recall))

# Applying the model to a new data point

print("\n\nApplying LR to the machine n°74...\n")
engine_number = 74
new_engine = test[test['id'] == engine_number]
#X_new_engine, Y_new_engine = flip_data(df=new_engine,
feature_columns=feature_columns, lookback_length=LOOKBACK_LENGTH )

Y_predicted_new_engine_lr = lr_model.predict(X_test_lr[X_test_lr['id'] ==
engine_number])

max_cycles = new_engine.shape[0]
cycles = range(LOOKBACK_LENGTH,(max_cycles-1),1)

new_engine_results = pd.DataFrame({'Cycle':cycles, 'LogisticR' :
Y_predicted_new_engine_lr[(LOOKBACK_LENGTH+1):]})

print(new_engine_results.tail(30))

# print the predicted failure day

result=new_engine_results[new_engine_results['LogisticR']==1].iloc[0]
if(result.empty):
  print('error')
else:
  print('Fail at day number',result['Cycle']+30,'.')

# make the user enter which engine number is to be predicted
input_1 = input('enter the machine number to predict : ')

engine_number = int(input_1)
new_engine = test[test['id'] == engine_number]
#X_new_engine, Y_new_engine = flip_data(df=new_engine,
feature_columns=feature_columns, lookback_length=LOOKBACK_LENGTH )
```

```
Y_predicted_new_engine_lr = lr_model.predict(X_test_lr[X_test_lr['id'] ==
engine_number])

max_cycles = new_engine.shape[0]
cycles = range(LOOKBACK_LENGTH,(max_cycles-1),1)

new_engine_results = pd.DataFrame({'Cycle':cycles, 'LogisticR' :
Y_predicted_new_engine_lr[(LOOKBACK_LENGTH+1):]})


print(new_engine_results.tail(30))

try:
  result=new_engine_results[new_engine_results['LogisticR']==1].iloc[0]

  print('It will fail at',result['Cycle']+30,'days.')
except:
  print('It will not fail at least within 30days.')
```