```
# Gerardo Herrera... ann: (1 capa oculta con 15 neuronas, activation = 'relu', epoch=100)


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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 5GB to the current directory (/kaggle/working/) that gets preserved
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of


# https://medium.com/@randerson112358/build-your-own-artificial-neural-network-using-pytho


#Load libraries
from keras.models import Sequential
from keras.layers import Dense
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')


import time


#sensor77 = pd.read_csv('../input/vombas/sensor_procesado.csv')
#sensor77 = pd.read_csv('../input/10ks25/s25balanced10k.csv')
#sensor77 = pd.read_csv('../input/28k-s24-balan-vombas/sensor2-ordenado_status_sin_broken_

sensor77 = pd.read_csv('/content/drive/My Drive/datasets/sensor2-ordenado_status_sin_broke



#Show the shape (number of rows & columns)
sensor77.shape
```

```
(28002, 27)
```

```
#Show the number of missing (NAN, NaN, na) data for each column
sensor77.isnull().sum()
```

```
Unnamed: 0            0
timestamp             0
sensor_00             0
sensor_01            30
sensor_02             0
sensor_03             0
sensor_04             0
sensor_11             0
sensor_14             0
sensor_16             0
sensor_17             0
sensor_18             0
sensor_19             0
sensor_20             0
sensor_21             0
sensor_22             0
sensor_23             0
sensor_25             0
sensor_26             0
sensor_27             0
sensor_28             0
sensor_30             0
sensor_31             0
sensor_44             3
sensor_50         14004
sensor_51          2996
machine_status        0
dtype: int64
```

```
cleanup_nums = {"machine_status":     {"NORMAL": 0, "RECOVERING": 1,"BROKEN": 2}}


sensor77.replace(cleanup_nums, inplace=True)


sensor77.fillna(sensor77.mean(), inplace=True)


#Show the number of missing (NAN, NaN, na) data for each column
sensor77.isnull().sum()
```

```
Unnamed: 0        0
timestamp         0
sensor_00         0
sensor_01         0
sensor_02         0
sensor_03         0
sensor_04         0
sensor_11         0
sensor_14         0
sensor_16         0
sensor_17         0
sensor_18         0
sensor_19         0
sensor_20         0
sensor_21         0
```

```
#sensor77.drop('sensor_15', axis=1, inplace=True)
sensor77.drop('timestamp', axis=1, inplace=True)
```

```
sensor_26         0
```

```
#sensor77.drop('100000', axis=1, inplace=True)
```

```
sensor_30         0
```

```
sensor77.drop('Unnamed: 0', axis=1, inplace=True)
```

```
sensor_50         0
```

```
sensor77.isnull().sum()
```

```
sensor_00         0
sensor_01         0
sensor_02         0
sensor_03         0
sensor_04         0
sensor_11         0
sensor_14         0
sensor_16         0
sensor_17         0
sensor_18         0
sensor_19         0
sensor_20         0
sensor_21         0
sensor_22         0
sensor_23         0
sensor_25         0
sensor_26         0
sensor_27         0
sensor_28         0
sensor_30         0
sensor_31         0
sensor_44         0
sensor_50         0
sensor_51         0
machine_status    0
dtype: int64
```

```
#Convert the data into an array
dataset = sensor77.values
dataset
```

```
array([[2.46539400e+00, 4.70920100e+01, 5.32118000e+01, ...,
        4.81174107e+02, 1.77951400e+02, 0.00000000e+00],
       [2.46539400e+00, 4.70920100e+01, 5.32118000e+01, ...,
        4.81174107e+02, 1.78530100e+02, 0.00000000e+00],
       [2.44473400e+00, 4.73524300e+01, 5.32118000e+01, ...,
        4.81174107e+02, 1.77662000e+05, 0.00000000e+00],
       ...,
       [2.40538200e+00, 4.95659714e+01, 5.38194400e+01, ...,
        3.21180573e+01, 3.15393524e+01, 1.00000000e+00],
       [2.40046300e+00, 4.95659700e+01, 5.37760400e+01, ...,
        3.21180573e+01, 3.15393500e+01, 1.00000000e+00],
       [2.40144700e+00, 4.95225700e+01, 5.37760391e+01, ...,
        3.21180573e+01, 3.18287000e+01, 1.00000000e+00]])
```

```
sensor77.shape
```

```
(28002, 25)
```

```
# Get all of the rows from the first eight columns of the dataset
#X = dataset[:,0:51]
X = dataset[:,0:24]
# Get all of the rows from the last column
#y = dataset[:,51]
y = dataset[:,24]
```

```
print(y)
```

```
[0. 0. 0. ... 1. 1. 1.]
```

```
print(X)
```

```
[[2.46539400e+00 4.70920100e+01 5.32118000e+01 ... 4.36921300e+01
  4.81174107e+02 1.77951400e+02]
 [2.46539400e+00 4.70920100e+01 5.32118000e+01 ... 4.45601800e+01
  4.81174107e+02 1.78530100e+02]
 [2.44473400e+00 4.73524300e+01 5.32118000e+01 ... 4.60069400e+01
  4.81174107e+02 1.77662000e+05]
 ...
 [2.40538200e+00 4.95659714e+01 5.38194400e+01 ... 3.15393524e+01
  3.21180573e+01 3.15393524e+01]
 [2.40046300e+00 4.95659700e+01 5.37760400e+01 ... 3.15393524e+01
  3.21180573e+01 3.15393500e+01]
 [2.40144700e+00 4.95225700e+01 5.37760391e+01 ... 3.15393524e+01
  3.21180573e+01 3.18287000e+01]]
```

```
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()
X_scale = min_max_scaler.fit_transform(X)
X_scale
```

```
      array([[1.16018541e-03, 1.51254935e-04, 2.97595181e-04, ...,
              7.32072243e-05, 1.01425222e-03, 3.84471811e-04],
             [1.16018541e-03, 1.51254935e-04, 2.97595181e-04, ...,
              7.67494867e-05, 1.01425222e-03, 3.85953388e-04],
             [1.15046306e-03, 1.56160565e-04, 2.97595181e-04, ...,
              8.26532983e-05, 1.01425222e-03, 4.54775949e-01],
             ...,
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scale, y, test_size=0.2, random_stat
```

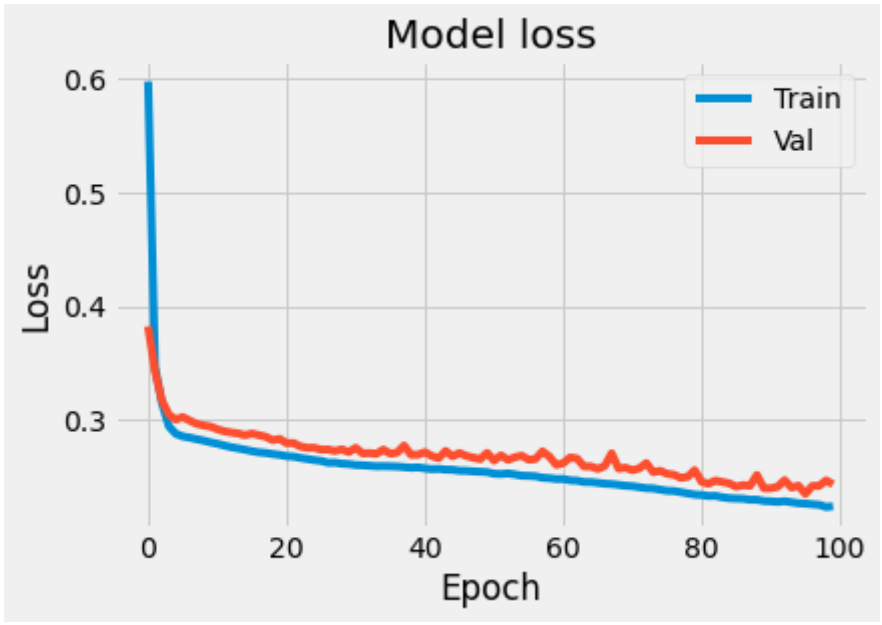
```
             [1.12962965e-03, 1.97857860e-04, 3.08223654e-04, ...,
```

```
model = Sequential([
    Dense(24, activation='relu', input_shape=( 24 ,)),
    #Dense(12, activation='relu', input_shape=( 51 ,)),
    Dense(15, activation='relu'),
    Dense(15, activation='relu'),
    Dense(15, activation='relu'),
    Dense(1, activation='sigmoid')
])


#model.compile(optimizer='sgd',
#              loss='binary_crossentropy',
#              metrics=['accuracy'])

model.compile(optimizer='sgd',
              loss='binary_crossentropy',
              metrics=['accuracy'])


start = time.time()
hist = model.fit(X_train, y_train,
          batch_size=10, epochs=100, validation_split=0.2)
stop = time.time()
print(f"Training time: {stop - start}s")
# prints: Training time: 0.20307230949401855s

# https://machinelearningmastery.com/evaluate-performance-deep-learning-models-keras/
```

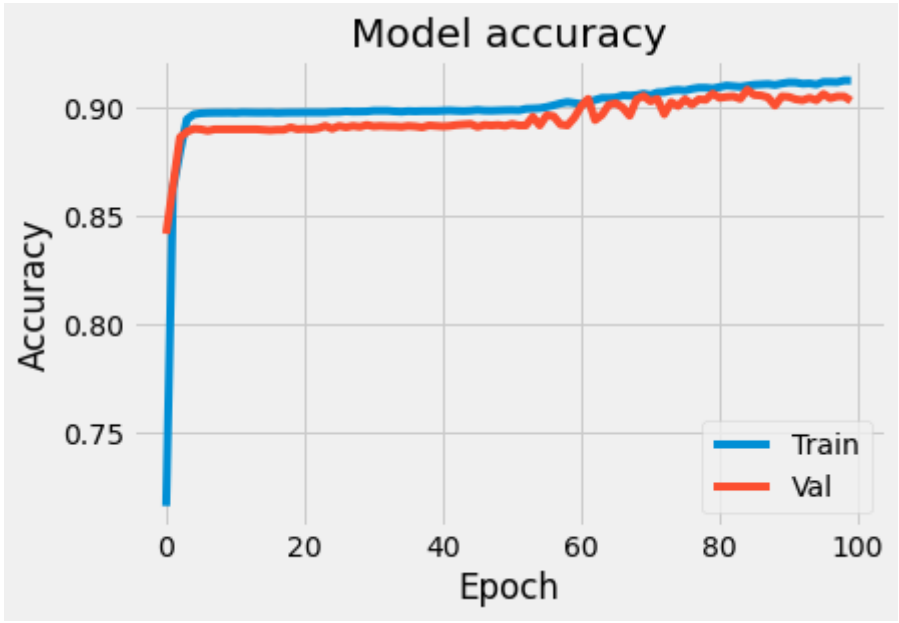
```
1792/1792 [==============================] - 2s 1ms/step - loss: 0.2405 - accuracy: 0
Epoch 73/100
1792/1792 [==============================] - 3s 2ms/step - loss: 0.2396 - accuracy: 0
Epoch 74/100
1792/1792 [==============================] - 3s 1ms/step - loss: 0.2394 - accuracy: 0
Epoch 75/100
1792/1792 [==============================] - 3s 1ms/step - loss: 0.2384 - accuracy: 0
Epoch 76/100
1792/1792 [==============================] - 2s 1ms/step - loss: 0.2374 - accuracy: 0
Epoch 77/100
1792/1792 [==============================] - 2s 1ms/step - loss: 0.2372 - accuracy: 0
Epoch 78/100
1792/1792 [==============================] - 3s 1ms/step - loss: 0.2361 - accuracy: 0
Epoch 79/100
1792/1792 [==============================] - 3s 2ms/step - loss: 0.2350 - accuracy: 0
Epoch 80/100
1792/1792 [==============================] - 2s 1ms/step - loss: 0.2340 - accuracy: 0
Epoch 81/100
1792/1792 [==============================] - 2s 1ms/step - loss: 0.2336 - accuracy: 0
Epoch 82/100
1792/1792 [==============================] - 3s 1ms/step - loss: 0.2327 - accuracy: 0
Epoch 83/100
1792/1792 [==============================] - 2s 1ms/step - loss: 0.2329 - accuracy: 0
Epoch 84/100
1792/1792 [==============================] - 3s 1ms/step - loss: 0.2316 - accuracy: 0
Epoch 85/100
1792/1792 [==============================] - 2s 1ms/step - loss: 0.2308 - accuracy: 0
Epoch 86/100
1792/1792 [==============================] - 2s 1ms/step - loss: 0.2305 - accuracy: 0
Epoch 87/100
1792/1792 [==============================] - 2s 1ms/step - loss: 0.2304 - accuracy: 0
Epoch 88/100
1792/1792 [==============================] - 2s 1ms/step - loss: 0.2295 - accuracy: 0
Epoch 89/100
1792/1792 [==============================] - 2s 1ms/step - loss: 0.2295 - accuracy: 0
Epoch 90/100
1792/1792 [==============================] - 2s 1ms/step - loss: 0.2283 - accuracy: 0
Epoch 91/100
1792/1792 [==============================] - 2s 1ms/step - loss: 0.2281 - accuracy: 0
Epoch 92/100
1792/1792 [==============================] - 2s 1ms/step - loss: 0.2276 - accuracy: 0
Epoch 93/100
1792/1792 [==============================] - 2s 1ms/step - loss: 0.2282 - accuracy: 0
Epoch 94/100
1792/1792 [==============================] - 2s 1ms/step - loss: 0.2272 - accuracy: 0
Epoch 95/100
1792/1792 [==============================] - 2s 1ms/step - loss: 0.2264 - accuracy: 0
Epoch 96/100
1792/1792 [==============================] - 2s 1ms/step - loss: 0.2261 - accuracy: 0
Epoch 97/100
1792/1792 [==============================] - 2s 1ms/step - loss: 0.2254 - accuracy: 0
Epoch 98/100
1792/1792 [==============================] - 2s 1ms/step - loss: 0.2250 - accuracy: 0
Epoch 99/100
1792/1792 [==============================] - 5s 3ms/step - loss: 0.2231 - accuracy: 0
Epoch 100/100
1792/1792 [==============================] - 4s 2ms/step - loss: 0.2238 - accuracy: 0
Training time: 261.8290750980377s
```

```
#visualize the training loss and the validation loss to see if the model is overfitting
```

```
#visualize the training loss and the validation loss to see if the model is overfitting
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```




```
#visualize the training accuracy and the validation accuracy to see if the model is overfi
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='lower right')
plt.show()
```




```
#Make a prediction & print the actual values
prediction = model.predict(X test)
```

```
#prediction  = [1 if y>=0.5 else 0 for y in prediction] #Threshold
prediction  = [1 if y>=0.75 else 0 for y in prediction] #Threshold
print(prediction)
print(y_test)
```

```
[1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0,
[1. 1. 0. ... 0. 0. 1.]
```

```
from sklearn.metrics import classification_report,confusion_matrix, accuracy_score
pred = model.predict(X_train)
#pred  = [1 if y>=0.5 else 0 for y in pred] #Threshold
pred  = [1 if y>=0.5 else 0 for y in pred] #Threshold
print(classification_report(y_train ,pred ))
print('Confusion Matrix: \n',confusion_matrix(y_train,pred))
print()
print('Accuracy: ', accuracy_score(y_train,pred))
print()
```

```
              precision    recall  f1-score   support

         0.0       0.85      0.99      0.92     11211
         1.0       0.99      0.83      0.90     11190

    accuracy                           0.91     22401
   macro avg       0.92      0.91      0.91     22401
weighted avg       0.92      0.91      0.91     22401

Confusion Matrix:
 [[11101   110]
 [ 1922  9268]]

Accuracy:  0.9092897638498282
```

```
from sklearn.metrics import classification_report,confusion_matrix, accuracy_score
pred = model.predict(X_test)
#pred  = [1 if y>=0.5 else 0 for y in pred] #Threshold
pred  = [1 if y>=0.5 else 0 for y in pred] #Threshold
print(classification_report(y_test ,pred ))
print('Confusion Matrix: \n',confusion_matrix(y_test,pred))
print()
print('Accuracy: ', accuracy_score(y_test,pred))
print()
```

```
                  precision    recall  f1-score   support

             0.0       0.85      0.99      0.91      2790
             1.0       0.99      0.82      0.90      2811
```

```
model.evaluate(X_test, y_test)[1]
```

```
 176/176 [==============================] - 0s 1ms/step - loss: 0.2393 - accuracy: 0.9
    0.9059096574783325
```

```
# ann cros vaidacion
# https://medium.com/datadriveninvestor/k-fold-and-dropout-in-artificial-neural-network-ea
```

```
#builing the neural net
from keras import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score


#accuracies =  cross_val_score(estimator=classifier, X= X, y=output_category,cv=10, n_jobs
#accuracies

#accuracies =  cross_val_score(estimator=model, X= X_test, y=pred,cv=5, n_jobs=-1)
#accuracies


# https://medium.com/analytics-vidhya/artificial-neural-network-ann-with-keras-simplified-


from sklearn.model_selection import cross_validate
from sklearn.model_selection import cross_val_score
def kera_classifier():
    cf = Sequential()
    cf.add(Dense(units = 12,  activation = 'relu', input_dim = 24))
    #cf.add(Dense(units = 12,  activation = 'relu', input_dim = 51))
    cf.add(Dense(units = 15,  activation = 'relu'))
    cf.add(Dense(units = 1, activation = 'sigmoid'))
    cf.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
    return cf
start7 = time.time()
cf = KerasClassifier(build_fn = kera_classifier, batch_size = 10, epochs = 100)
#cf = KerasClassifier(build_fn = kera_classifier, batch_size = 57, epochs = 100)
#acuracies = cross_val_score(estimator = cf, X = X_train, y = y_train, cv = 10, n_jobs = -
#accuracies = cross_validate(cf, X = X_train, y = y_train, cv = 10, n_jobs = -1,scoring ='
accuracies = cross_val_score(cf, X = X_train, y = y_train, cv = 10, n_jobs = -1,scoring ='
#ean = accuracies.mean()
#iance = accuracies.std()
#
prec = cross_val_score(cf, X = X_train, y = y_train, cv = 10, n_jobs = -1 ,scoring ='preci
f1 = cross_val_score(cf, X = X_train, y = y_train, cv = 10, n_jobs = -1 ,scoring ='f1')
recal = cross_val_score(cf, X = X_train, y = y_train, cv = 10, n_jobs = -1 ,scoring ='reca
#ean1= recal.mean()
#ariance1= recal.std()
```