TP_Mantenimiento_Predictivo_Adrian_Cafa

July 15, 2020

1 Mantenimiento predictivo: detección de fallos de sensores con CNN

Adrián Pablo Cafa - ANALISIS Y VISUALIZACION DE DATOS CIENTIFICOS Y GEOGRAFICOS

EJERCICIO FINAL PUNTO 3: Busquen un dataset de internet público de señales de sensores. ¿Cómo lo abordarían exploratoriamente, qué procesamiento y qué análisis harían?'

Se eligió analizar un problema de clasificación. Se utilizará un modelo hecho con la CNN en Keras:

https://archive.ics.uci.edu/ml/datasets/Condition+monitoring+of+hydraulic+systems

Este dataset es un conjunto de datos de la UCI (Condition Monitoring of Hydraulic Systems). El mismo que se describirá a continuación, se obtuvo experimentalmente con un equipo de pruebas hidráulicas. Este equipo de prueba consiste en un circuito primario de trabajo y un circuito secundario de refrigeración y filtración que están conectados a través del tanque de aceite. El sistema repite cíclicamente ciclos de carga constantes (de duración 60 segundos) y mide valores de proceso como presiones, flujos de volumen y temperaturas, mientras que el estado de los cuatro componentes hidráulicos (refrigerador, válvula, bomba y acumulador) se varía cuantitativamente. Se puede imaginar tener un sistema de tuberías hidráulicas que recibe cíclicamente impulsos debido, por ejemplo, a la transición de un determinado tipo de líquido en la tubería. Este fenómeno dura 60 segundos y fue medido por diferentes sensores (Sensor Physical quantity Unit Sampling rate, PS1 Pressure bar, PS2 Pressure bar, PS3 Pressure bar, PS4 Pressure bar, PS5 Pressure bar, PS6 Pressure bar, EPS1 Motor power, FS1 Volume flow, FS2 Volume flow, TS1 Temperature, TS2 Temperature, TS3 Temperature, TS4 Temperature, VS1 Vibration, CE Cooling efficiency, CP Cooling power, SE Efficiency factor) con diferentes frecuencias [Hz].

> El objetivo o análisis es: predecir el estado de los cuatro componentes hidráulicos que componen la tubería. Estos valores de condición objetivo se anotan en forma de valores enteros (esto es fácil de codificar) y nos dicen si un componente en particular está a punto de fallar en cada ciclo. Los valores medidos por cada sensor están disponibles en un archivo txt, en el que cada row representa un ciclo en forma de serie temporal.

IMPORTANTE: Se decidió tener en cuenta los datos procedentes de los sensores de temperatura (TS1, TS2, TS3, TS4) medidos con una frecuencia de 1 Hz (60 observaciones por cada ciclo).

2 Descripción del Dataset:

Sacado propiamente de UCI:

The data set contains raw process sensor data (i.e. without feature extraction) which are structured as matrices (tab-delimited) with the rows representing the cycles and the columns the data points within a cycle. The sensors involved are:

Sensor	Physical quantity	Unit	Sampling rate
PS1	Pressure	bar	100 Hz
PS2	Pressure	bar	100 Hz
PS3	Pressure	bar	100 Hz
PS4	Pressure	bar	100 Hz
PS5	Pressure	bar	100 Hz
PS6	Pressure	bar	100 Hz
EPS1	Motor power	W	100 Hz
FS1	Volume flow	l/min	10 Hz
FS2	Volume flow	l/min	10 Hz
TS1	Temperature	°C	1 Hz
TS2	Temperature	°C	1 Hz
TS3	Temperature	°C	1 Hz
TS4	Temperature	°C	1 Hz
VS1	Vibration	mm/s	1 Hz
CE	Cooling efficiency (virtual)	%	1 Hz
CP	Cooling power (virtual)	kW	1 Hz
SE	Efficiency factor	%	1 Hz

The target condition values are cycle-wise annotated in ‘profile.txt‘ (tab-delimited). As before, the row number represents the cycle number. The columns are

```
1: Cooler condition / %:
    3: close to total failure
    20: reduced effifiency
    100: full efficiency

2: Valve condition / %:
    100: optimal switching behavior
    90: small lag
    80: severe lag
    73: close to total failure

3: Internal pump leakage:
    0: no leakage
    1: weak leakage
    2: severe leakage

4: Hydraulic accumulator / bar:
    130: optimal pressure
    115: slightly reduced pressure
    100: severely reduced pressure
    90: close to total failure
```

```
5: stable flag:
    0: conditions were stable
    1: static conditions might not have been reached yet
```

[25]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import itertools
import random
import os

from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report
from sklearn.model_selection import train_test_split
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA

import tensorflow as tf
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from tensorflow.keras.utils import *
```

[2]:
```
### LECTURA DE LOS DATOS ###

label = pd.read_csv('profile.txt', sep='\t', header=None)
label.columns = ['Cooler','Valve','Pump','Accumulator','Flag']

data = ['TS1.txt','TS2.txt','TS3.txt','TS4.txt']
df = pd.DataFrame()

for txt in data:
    read_df = pd.read_csv(txt, sep='\t', header=None)
    df = df.append(read_df)

print(df.shape)
df.head()
```

```
(8820, 60)
```

[2]:

	0	1	2	3	4	5	6	7	8	\
0	35.570	35.492	35.469	35.422	35.414	35.320	35.227	35.242	35.160	
1	36.156	36.094	35.992	36.008	35.992	35.902	35.824	35.820	35.727	
2	37.488	37.391	37.340	37.312	37.223	37.145	37.059	36.973	36.898	
3	38.633	38.535	38.469	38.379	38.297	38.223	38.125	38.062	37.977	
4	39.461	39.461	39.375	39.281	39.203	39.113	39.043	38.969	38.875	

	9	…	50	51	52	53	54	55	56	\

```
0  35.176  …  36.008  35.984  35.996  36.039  36.008  36.008  36.094
1  35.727  …  37.328  37.324  37.340  37.332  37.316  37.410  37.418
2  36.879  …  38.457  38.461  38.457  38.469  38.469  38.555  38.527
3  37.969  …  39.441  39.363  39.367  39.457  39.461  39.461  39.473
4  38.883  …  40.324  40.320  40.312  40.340  40.320  40.387  40.391

       57      58      59
0  36.102  36.090  36.152
1  37.422  37.488  37.477
2  38.543  38.527  38.621
3  39.441  39.453  39.461
4  40.391  40.387  40.391

[5 rows x 60 columns]
```

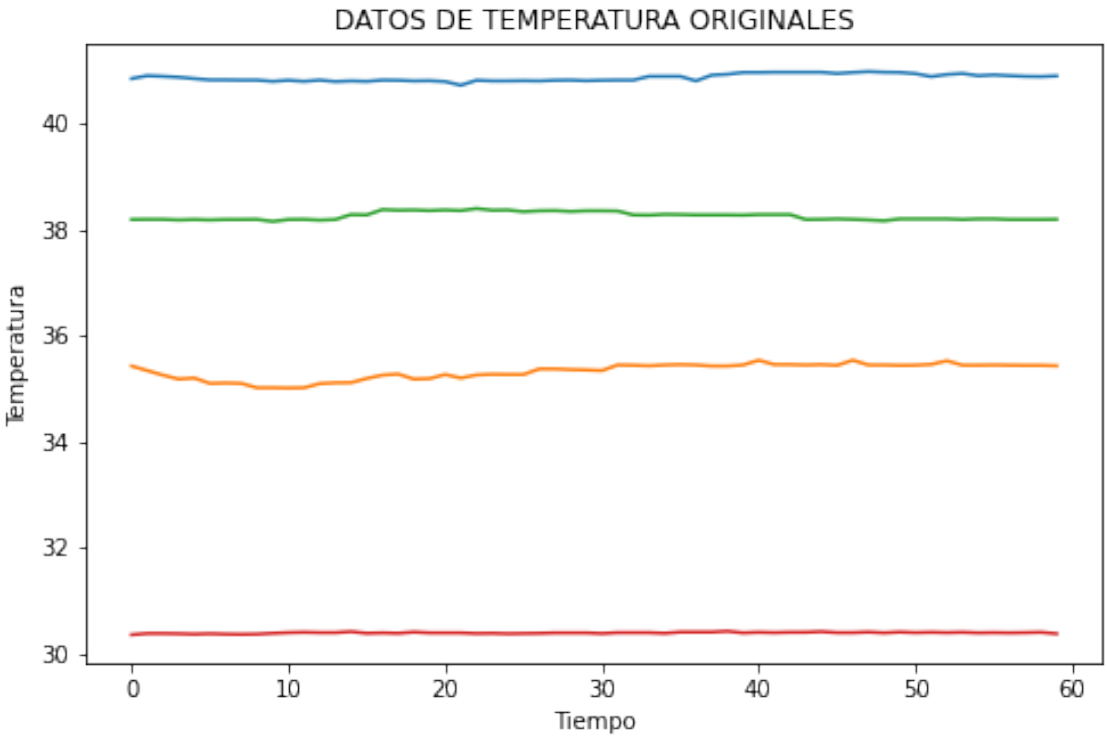
[3]:
```
label.shape
```

[3]: (2205, 5)

[4]:
```
### REFORMULACION DE LOS DATOS ENTRE LOS SENSORES ###

df = df.sort_index().values.reshape(-1,len(data),len(df.columns)).
 ↪transpose(0,2,1)
df.shape
```

[4]: (2205, 60, 4)

[5]:
```
### PLOT DE LA TEMPERATURA ORIGINAL PARA EL ULTIMO CICLO ###

plt.figure(figsize=(8,5))
plt.plot(df[2204])
plt.title('DATOS DE TEMPERATURA ORIGINALES')
plt.ylabel('Temperatura'); plt.xlabel('Tiempo')
np.set_printoptions(False)
```



[6]:
```
### DISTRIBUCION DE LOS LABELS ###

label = label.Cooler
label.value_counts()
```

[6]:
```
100    741
3      732
20     732
Name: Cooler, dtype: int64
```

En nuestro estudio nos interesa “Cooler condition”

[7]:
```
label.tail()
```

[7]:
```
2200    100
2201    100
2202    100
2203    100
2204    100
Name: Cooler, dtype: int64
```

[8]:
```
### MAPEO DE LOS LABEL ###
# Hago un diccionario para identificar los targets
```

```
diz_label, diz_reverse_label = {}, {}
for i,lab in enumerate(label.unique()):
    diz_label[lab] = i
    diz_reverse_label[i] = lab

print(diz_label)
print(diz_reverse_label)
label = label.map(diz_label)
y = to_categorical(label)
```

```
{3: 0, 20: 1, 100: 2}
{0: 3, 1: 20, 2: 100}
```

[9]:
```
y.shape
```

[9]:
```
(2205, 3)
```

[10]:
```
label.tail()
```

[10]:
```
2200    2
2201    2
2202    2
2203    2
2204    2
Name: Cooler, dtype: int64
```

[11]:
```
### TRAIN TEST SPLIT ###

X_train, X_test, y_train, y_test = train_test_split(df, y, random_state = 42,␣
 ↪test_size=0.2)
```

[12]:
```
### SCALE DATA ###

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train.reshape(-1, X_train.shape[-1])).
 ↪reshape(X_train.shape)
X_test = scaler.transform(X_test.reshape(-1, X_test.shape[-1])).reshape(X_test.
 ↪shape)
```

[13]:
```
X_train.shape
```

[13]:
```
(1764, 60, 4)
```

[14]:
```
X_test.shape
```

[14]: (441, 60, 4)

3 Creación del Modelo

Con el fin de captar características interesantes y correlaciones no obvias de las series, se adoptó una CNN 1D. Este tipo de modelo se adapta muy bien al análisis de las secuencias temporales de los sensores. Se utiliza la misma CNN que se describe en el siguiente sitio web de Keras: https://keras.io/guides/sequential_model/ El modelo se construyó para clasificar el estado del componente de enfriamiento (Cooler condition) dando como entrada sólo la serie de tiempo de la temperatura en formato array (t_períodos x n_sensor para cada ciclo individual).

Sacado del siguiente artículo:

https://machinelearningmastery.com/cnn-models-for-human-activity-recognition-time-series-classification/

Cita:

Los modelos de redes neuronales convolutivas se desarrollaron para problemas de clasificación de imágenes, en los que el modelo aprende una representación interna de una entrada bidimensional, en un proceso denominado aprendizaje de características. Este mismo proceso puede aprovecharse en secuencias unidimensionales de datos. El modelo aprende a extraer características de las secuencias de observaciones y a mapear las características internas a los diferentes tipos de actividad.

La ventaja de utilizar las CNN para la clasificación de secuencias es que pueden aprender directamente de los datos brutos de las series temporales y, a su vez, no requieren conocimientos especializados para diseñar manualmente las características de entrada. El modelo puede aprender una representación interna de los datos de las series cronológicas e idealmente lograr un rendimiento comparable al de los modelos que se ajustan a una versión del conjunto de datos con características de ingeniería.

Se tomo de referencia el siguiente artículo también:

https://blog.goodaudience.com/introduction-to-1d-convolutional-neural-networks-in-keras-for-time-sequences-3a7ff801a2cf

[15]:
```
num_sensors = 4
TIME_PERIODS = 60
BATCH_SIZE = 16
EPOCHS = 10

model_m = Sequential()
model_m.add(Conv1D(100, 6, activation='relu', input_shape=(TIME_PERIODS,␣
 ↪num_sensors)))
model_m.add(Conv1D(100, 6, activation='relu'))
model_m.add(MaxPooling1D(3))
model_m.add(Conv1D(160, 6, activation='relu'))
model_m.add(Conv1D(160, 6, activation='relu'))
model_m.add(GlobalAveragePooling1D(name='G_A_P_1D'))
model_m.add(Dropout(0.5))
```

```
model_m.add(Dense(3, activation='softmax'))

model_m.compile(loss='categorical_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])
history = model_m.fit(X_train, y_train, batch_size=BATCH_SIZE, epochs=EPOCHS,␣
↪validation_split=0.2, verbose=2)
```

```
Train on 1411 samples, validate on 353 samples
Epoch 1/10
1411/1411 - 2s - loss: 0.2012 - accuracy: 0.9504 - val_loss: 0.0812 -
val_accuracy: 0.9802
Epoch 2/10
1411/1411 - 1s - loss: 0.1199 - accuracy: 0.9780 - val_loss: 0.0795 -
val_accuracy: 0.9858
Epoch 3/10
1411/1411 - 1s - loss: 0.1034 - accuracy: 0.9752 - val_loss: 0.0722 -
val_accuracy: 0.9802
Epoch 4/10
1411/1411 - 1s - loss: 0.1070 - accuracy: 0.9745 - val_loss: 0.0818 -
val_accuracy: 0.9802
Epoch 5/10
1411/1411 - 1s - loss: 0.1145 - accuracy: 0.9752 - val_loss: 0.0676 -
val_accuracy: 0.9858
Epoch 6/10
1411/1411 - 1s - loss: 0.1076 - accuracy: 0.9759 - val_loss: 0.0685 -
val_accuracy: 0.9830
Epoch 7/10
1411/1411 - 1s - loss: 0.0923 - accuracy: 0.9780 - val_loss: 0.0679 -
val_accuracy: 0.9830
Epoch 8/10
1411/1411 - 1s - loss: 0.0939 - accuracy: 0.9794 - val_loss: 0.0544 -
val_accuracy: 0.9858
Epoch 9/10
1411/1411 - 1s - loss: 0.1063 - accuracy: 0.9759 - val_loss: 0.0670 -
val_accuracy: 0.9802
Epoch 10/10
1411/1411 - 1s - loss: 0.0849 - accuracy: 0.9809 - val_loss: 0.0580 -
val_accuracy: 0.9858
```

Se verifico que el modelo con 10 epoch alcanza un accuracy de aproximadamente el 97% para los datos de entrenamiento.

[16]:
```
model_m.evaluate(X_test, y_test, verbose=2)
```

```
441/441 - 0s - loss: 0.0411 - accuracy: 0.9932
```

[16]:
```
[0.0411265689220761, 0.99319726]
```

[17]:
```
### OBTENCION DE LA CLASE A PREDECIR ###

pred_test = np.argmax(model_m.predict(X_test), axis=1)
```

[18]:
```
print(classification_report([diz_reverse_label[np.argmax(label)] for label in␣
↪y_test],
                            [diz_reverse_label[label] for label in pred_test]))
```

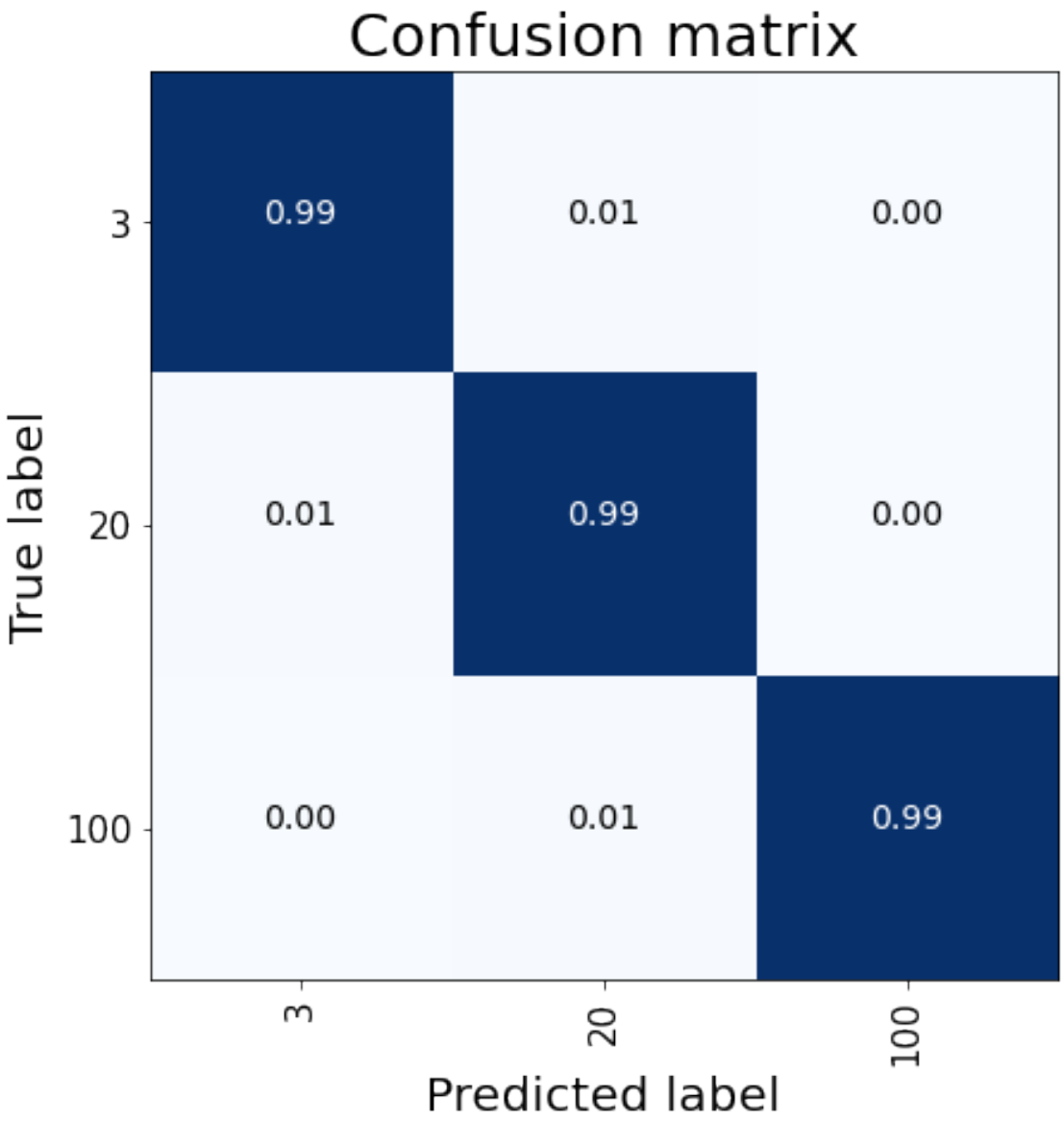
```
              precision    recall  f1-score   support

           3       0.99      0.99      0.99       152
          20       0.99      0.99      0.99       135
         100       1.00      0.99      1.00       154

    accuracy                           0.99       441
   macro avg       0.99      0.99      0.99       441
weighted avg       0.99      0.99      0.99       441
```

Ahora se graficará la matriz de confusion del modelo

[19]:
```
def plot_confusion_matrix(cm, classes, title='Confusion matrix', cmap=plt.cm.
↪Blues):

    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title, fontsize=25)
    #plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90, fontsize=15)
    plt.yticks(tick_marks, classes, fontsize=15)

    fmt = '.2f'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black", fontsize = 14)

    plt.ylabel('True label', fontsize=20)
    plt.xlabel('Predicted label', fontsize=20)
```

[20]:
```
cnf_matrix = confusion_matrix([diz_reverse_label[np.argmax(label)] for label in␣
↪y_test],
                              [diz_reverse_label[label] for label in pred_test])

plt.figure(figsize=(7,7))
```

```
plot_confusion_matrix(cnf_matrix, classes=list(diz_reverse_label.values()))
plt.show()
```



Se verifico que haciendo la predicción en los datos de test el modelo alcanza una EXACTITUD del 0,9909%.

IMPORTANTE: Este resultado es útil para la clase 3 (del componente Cooler condition "cercano a la falla total") porque de esta manera podemos detectar y prevenir posibles fallas en el sistema.

4 Representación visual:

Se reutiliza la CNN para hacer un decodificador y extraer características inteligentes de la serie temporal de cada ciclo. Se tomo de ejemplo Con Keras se puede realizar en una sola línea de código:

[21]:
```
emb_model = Model(inputs=model_m.input, outputs=model_m.get_layer('G_A_P_1D').
↪output)
emb_model.summary()
```

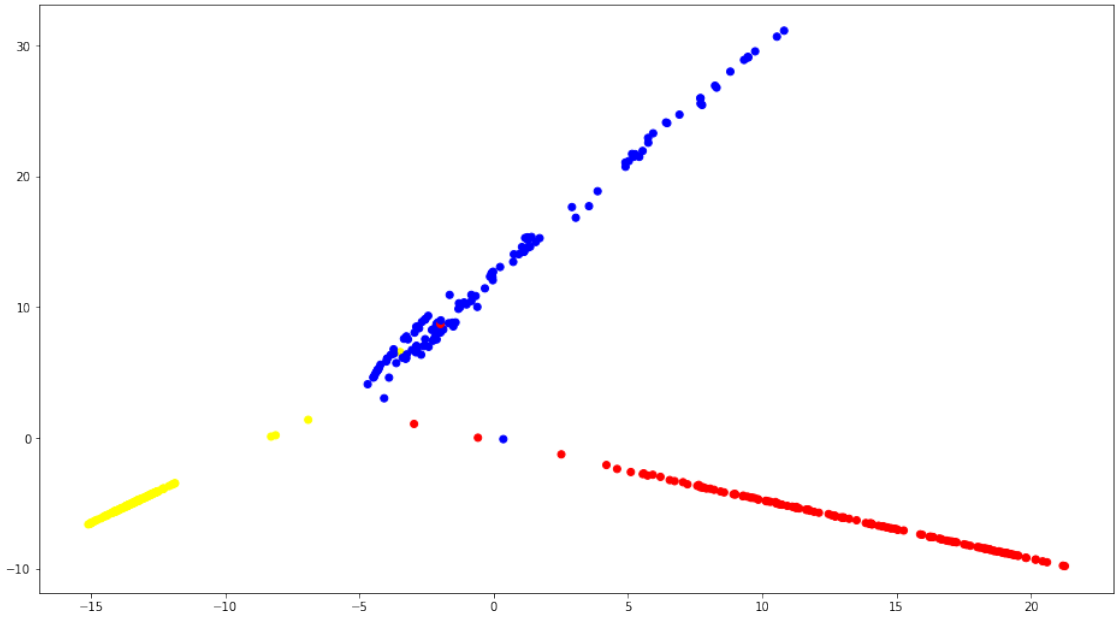
```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_input (InputLayer)    [(None, 60, 4)]           0
_________________________________________________________________
conv1d (Conv1D)              (None, 55, 100)           2500
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 50, 100)           60100
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 16, 100)           0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 11, 160)           96160
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 6, 160)            153760
_________________________________________________________________
G_A_P_1D (GlobalAveragePooli (None, 160)               0
=================================================================
Total params: 312,520
Trainable params: 312,520
Non-trainable params: 0
_________________________________________________________________
```

El nuevo modelo es un decodificador que recibe como entrada datos en el mismo formato de la NN que se utilizó para la tarea de clasificación (t_períodos x n_sensor para cada ciclo), es decir se toma la red YA ENTRENADA; y devuelve la "predicción" en forma de embeddings procedentes de la capa GlobalAveragePooling1D con dimensión relativa (una fila de 160 variables de embeddings para cada ciclo). Computando la predicción con nuestro codificador en los datos de test, adoptando una técnica para reducir las dimensiones (como PCA o T-SNE) y ploteando los resultados podemos ver lo siguiente, se selecciono PCA por tener experiencia usandolo

[22]:
```
### EXTRACCION DE EMBEDDINGS ###
serie_features = emb_model.predict(X_test)
```

[23]:
```
### VISUALIZACION DE EMBEDDINGS ###
pca = PCA(n_components=2)
T = pca.fit_transform(serie_features)
```

[24]:
```
plt.figure(figsize=(16,9))
colors = {0:'red', 1:'blue', 2:'yellow'}
plt.scatter(T.T[0], T.T[1], c=[colors[i] for i in np.argmax(y_test, axis=1)])
plt.show()
```



Cada punto representa un ciclo en el conjunto de test y el color relativo es la clase objetivo de la condición de cooler. Es posible ver cómo la distinción entre los valores objetivo del componente de "Cooler Condition" está bien definido. Se logró observar el poder que poseen las CNN no sólo en caso de predicción sino también como instrumento para detectar relaciones invisibles entre los datos.

```
Recordando del mapeo realizado:
{3: 0, 20: 1, 100: 2}

Siendo en este caso:

0:'red', 1:'blue', 2:'yellow'

Cooler condition / %:
    3: close to total failure (puntos rojos)
    20: reduced effifiency (puntos azules)
    100: full efficiency (puntos amarillos)
```