Practico Mentoria - Introduccion al Aprendizaje Automatico

Integrantes: Trapaglia Matias - Medrano Nazareno - Fontana Fernando

Se propone la elaboración de un informe o presentación, en formato estatico:

- PDF
- Markdowns
- Google Docs

Que responda a las cuestiones solicitadas en cada seccion de esta **Jupyter Notebook**.

La comunicación debe estar apuntada a un público técnico pero sin conocimiento del tema particular, como por ejemplo, sus compañeros de clase.
Por lo cual debe estar redactado de forma consisa y comprensible.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, mean_squared_error, confusion_matrix

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import accuracy_score, mean_squared_error, confusion_matri
x, classification_report
from sklearn import preprocessing
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
```

In [2]:
```
filename = "../Introduccion_Mentoria/dataset_inf_telec_20200501T130000_20200727T0
10000_v1.csv"
```

In [3]:
```
df = pd.read_csv(
    filename,
    dtype={
        'ID_EQUIPO': int,
        'PUNTO_MEDICION': str,
        'CAPACIDAD_MAXIMA_GBS': float,
        'PASO': int,
        'LATENCIA_MS': float,
        'PORCENTAJE_PACK_LOSS': float,
        'INBOUND_BITS': np.float64,
        'OUTBOUND_BITS': np.float64,
        'MEDIDA': str,
    },
    parse_dates=[
        'FECHA_INICIO_MEDICION',
        'FECHA_HORA',
        'FECHA_FIN_MEDICION',
    ],
    na_values=['NaN']
)
```

In [4]:
```
df.shape
```

Out[4]:
```
(61548, 12)
```

In [5]:
```
print(df.FECHA_HORA.min())

print(df.FECHA_HORA.max())
```

```
2020-05-01 03:00:00
2020-07-27 01:00:00
```

Regresion

Elija algun `PUNTO_MEDICION` , por ejemplo **ABA - Abasto Cliente**.

Vamos a predecir el `LATENCIA_MS` de dicho punto de medición.

In [6]:
```
df_orig = df.copy()
```

In [7]:
```
df= df_orig.copy()
```

In [8]:
```
df_orig.PUNTO_MEDICION.unique()
```

Out[8]:
```
array(['ABA - Abasto Cliente', 'ABA - Temple', 'BAZ - Carlos Paz',
       'BAZ - Yocsina', 'Carlos Paz - Cosquin', 'Carlos Paz - La Falda',
       'EDC - Capitalinas', 'EDC - Coral State', 'EDC - ET Oeste',
       'EDC - MOP', 'EDC - NOR', 'EDC - RDB', 'EDC - Telecomunicacioes',
       'EDC - Transporte', 'JM - Totoral Nueva', 'JM - Totoral Vieja',
       'NOC - 6720HI to BAZ', 'NOC - 6720HI to EDC',
       'NOC - 6720HI to ETC', 'NOC - 6720HI to N20-1',
       'NOC - 6720HI to R4 Silica', 'NOC - 6720HI to RPrivado',
       'NOC - ACHALA - Servicios', 'NOC - ACHALA - Solo Dolores',
       'NOC - Almacenes', 'NOC - ET Sur', 'NOC - Interfabricas',
       'NOC - Pilar', 'NOC - S9306 to SS6720HI', 'NOC - SW Clientes 1',
       'NOC - SW Clientes 2', 'NOC - Switch Servers', 'NOC - UTN',
       'RDB - Escuela de Capacitacion', 'RDB - ET Don Bosco - San Roque',
       'RDB - ET La Calera', 'RDB - GZU', 'RDB - JM', 'RDB - PEA',
       'RDB - RIO', 'SF - Freyre', 'SF - La Francia', 'SF - Las Varillas',
       'SF - SF Adm', 'SF - SF Cliente', 'Yocsina - Alta Gracia',
       'Yocsina - Carlos Paz', 'Yocsina - Mogote'], dtype=object)
```

In [9]:
```
df= df_orig.loc[df_orig.PUNTO_MEDICION=='BAZ - Carlos Paz',:]
```

In [10]:
```
df.shape
```

Out[10]:
```
(1291, 12)
```

In [11]:
```
df = df.dropna(subset=['LATENCIA_MS', 'OUTBOUND_BITS', 'INBOUND_BITS', 'PORCENTAJE_PACK_LOSS'])
```

In [12]:
```
# Separamos el "target" del resto del dataset
X = df.loc[:, df.columns != 'LATENCIA_MS']
y = df['LATENCIA_MS']
```

In [13]:
```
print(X.shape, y.shape)
```

```
(1290, 11) (1290,)
```

Seleccionamos uno o más feature del dataset que no sea categórico, por ejemplo `INBOUND_BITS`

In [14]:
```
# TODO: modificar esta feature por algún otro (o una combinacion de estos) para ver como cambian los resultados
X = X[['OUTBOUND_BITS']]
```

División de datos en conjuntos de entrenamiento y evaluación

La primer tarea consiste en dividir el conjunto de datos cargados en el apartado anterior en conjuntos de entrenamiento (training) y evaluación (test).

Utilizar aproximadamente 70% de los datos para entrenamiento y 30% para validación.

Links:

- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)

In [15]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=77)
```

In [16]:
```
X_train.sample(5)
```

Out[16]:

	OUTBOUND_BITS
42988	3.831258e+08
32382	3.911073e+08
16666	6.544469e+08
715	6.081352e+08
864	4.029404e+08

Regresion Lineal

Link:

- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)

In [17]:
```
scaler = preprocessing.StandardScaler()
model = LinearRegression()
pipe = Pipeline(steps=[("scaler", scaler), ("linear_reg", model)])
pipe.fit(X_train, y_train)
```

Out[17]:
```
Pipeline(steps=[('scaler', StandardScaler()),
                ('linear_reg', LinearRegression())])
```

Evaluamos el desempeño del clasificador utilizando la media del error cuadrado (`MSE` o `Mean Squared Error`) sobre el conjunto de datos de entrenamiento `(X_train, y_train)` y lo comparamos con el de validación `(X_val, y_test)` . Mientras más cercano a cero mejor

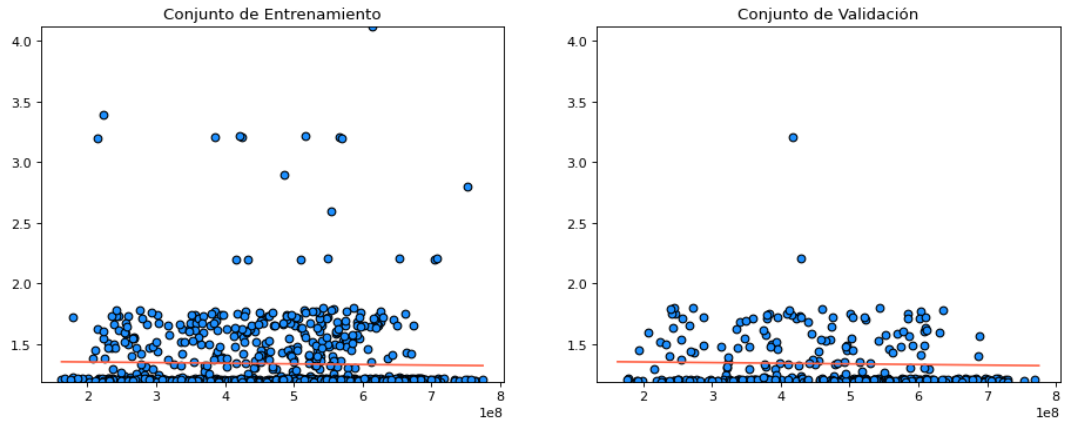
In [18]:
```
print(f"MSE para entrenamiento: {mean_squared_error(y_train, pipe.predict(X_train)):.2f}")
print(f"MSE para validación   : {mean_squared_error(y_test, pipe.predict(X_test)):.2f}")
```

```
MSE para entrenamiento: 0.09
MSE para validación   : 0.04
```

Visualizacion

Warning: Tener en cuenta que si son dos o mas features no se va a poder visualizar

In [19]:
```
plt.figure(figsize=(14, 5), dpi= 80, facecolor='w', edgecolor='k')

X_range_start = np.min(np.r_[X_train, X_test])
X_range_stop = np.max(np.r_[X_train, X_test])
y_range_start = np.min(np.r_[y_train, y_test])
y_range_stop = np.max(np.r_[y_train, y_test])
X_linspace = np.linspace(X_range_start, X_range_stop, 200).reshape(-1, 1)

# Conjunto de entrenamiento
plt.subplot(1, 2, 1)
plt.scatter(X_train, y_train, facecolor="dodgerblue", edgecolor="k", label="datos")
plt.plot(X_linspace, pipe.predict(X_linspace), color="tomato", label="modelo")
plt.ylim(y_range_start, y_range_stop)
plt.title("Conjunto de Entrenamiento")

# Conjunto de validación
plt.subplot(1, 2, 2)
plt.scatter(X_test, y_test, facecolor="dodgerblue", edgecolor="k", label="datos")
plt.plot(X_linspace, pipe.predict(X_linspace), color="tomato", label="modelo")
plt.ylim(y_range_start, y_range_stop)
plt.title("Conjunto de Validación")

plt.show()
```



Se puede apreciar que logramos un buen modelo, pero al ser este lineal, no se obserba mucha dependencia entre las features ananalizadas y la variable Latencia objetivo.

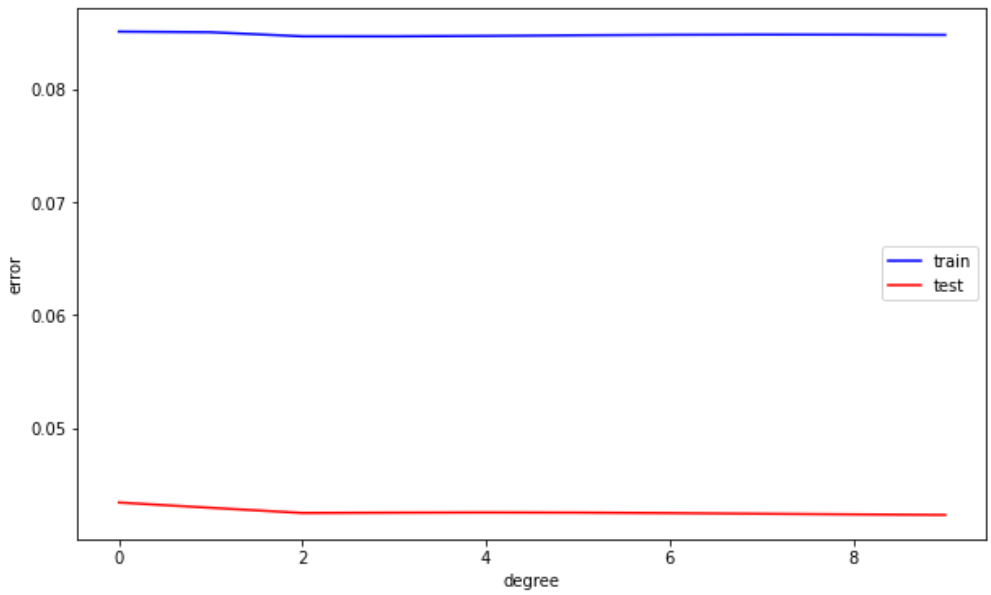
EXTRA: Regresión Polinomial

Link:

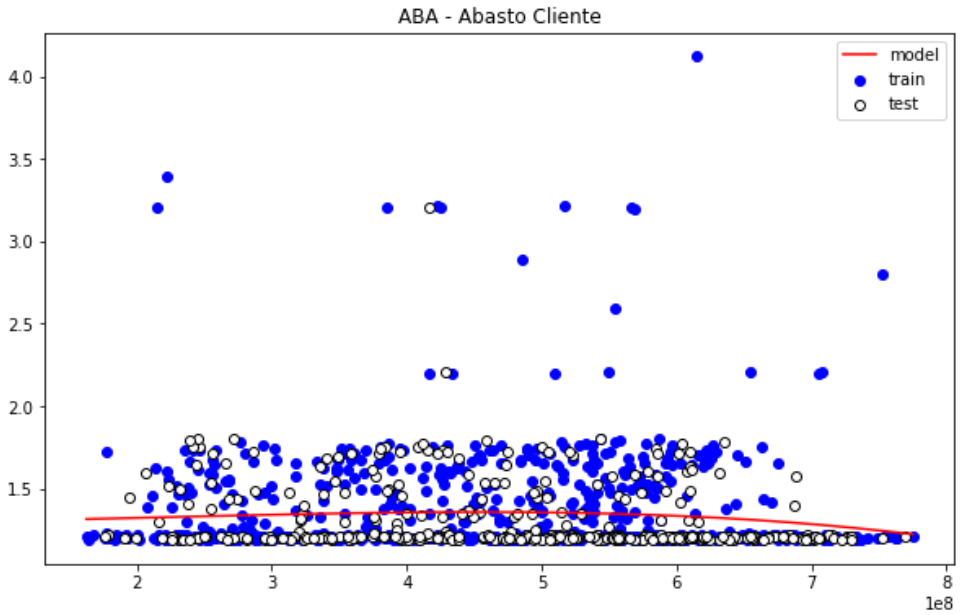
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html)

In [20]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

train_errors = []
test_errors = []

degrees = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

for degree in degrees:
    # train:
    model = make_pipeline(PolynomialFeatures(degree), LinearRegression())
    model.fit(X_train, y_train)

    # predict:
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    # evaluate:
    train_error = mean_squared_error(y_train, y_train_pred)
    test_error = mean_squared_error(y_test, y_test_pred)
    train_errors.append(train_error)
    test_errors.append(test_error)


print(f'\nMin train error: {min(train_errors)}')
print('Polinomio grado ' + str(train_errors.index(min(train_errors))),"\n")
print(f'Min test error: {min(test_errors)}')
print('Polinomio grado ' + str(test_errors.index(min(test_errors))),"\n")

fig = plt.figure(figsize=(10,6))
plt.plot(degrees, train_errors, color="blue", label="train")
plt.plot(degrees, test_errors, color="red", label="test")
plt.legend()
plt.xlabel("degree")
plt.ylabel("error")
plt.show()
```

```
Min train error: 0.08468803813724866
Polinomio grado 3

Min test error: 0.04229873622011721
Polinomio grado 9
```



In [21]:
```
degree = 3
model = make_pipeline(PolynomialFeatures(degree), LinearRegression())
model.fit(X_train, y_train);

fig = plt.figure(figsize=(10,6))
plt.scatter(X_train, y_train, color="blue", label="train")
plt.scatter(X_test, y_test, color="white", edgecolor="k", label="test")
plt.plot(X_linspace, model.predict(X_linspace.reshape(-1, 1)), color="red", labe
l="model")
plt.title('ABA - Abasto Cliente')
plt.legend()
plt.show()
```



Podemos apreciar que el modelo se adapta mejor a la forma que tiene el cúmulo de datos. Sin embargo esta aproximacion sigue sin ser exacta debido a la falta de correlacion entre features.

Clasificacion

Elija algun `PUNTO MEDICION` , por ejemplo **ABA - Abasto Cliente**.

Vamos a predecir el `PORCENTAJE_PACK_LOSS` de dicho punto de medición.
Como esta variable no es categorica, vamos a codificarla como tal (guiarse por lo que saben de **Analisis y Curacion de Datos**), para ello vamos a tomar los siguientes rangos:

- Si `PORCENTAJE_PACK_LOSS` $\in [0, 0.05) \Rightarrow$ 0
- Si `PORCENTAJE_PACK_LOSS` $\in [0.05, 0.1) \Rightarrow$ 1
- Si `PORCENTAJE_PACK_LOSS` $\in [0.1, \infty) \Rightarrow$ 2

In [22]:
```
df_orig[df_orig.PUNTO_MEDICION=='ABA - Abasto Cliente'].PORCENTAJE_PACK_LOSS.des
cribe()
```

Out[22]:
```
count    1290.0
mean        0.0
std         0.0
min         0.0
25%         0.0
50%         0.0
75%         0.0
max         0.0
Name: PORCENTAJE_PACK_LOSS, dtype: float64
```

No puedo tomar 'ABA - Abasto Cliente' porque el valor de packet loss es fijo para todas las muestras.

In [23]:
```
df_orig.PORCENTAJE_PACK_LOSS.describe()[-1]
```

Out[23]:
```
58.28972430108815
```

In [24]:
```
maximo = 0
for punto in df_orig.PUNTO_MEDICION.unique():
    media = df_orig[df_orig.PUNTO_MEDICION==punto].PORCENTAJE_PACK_LOSS.describe
()[1]
    maximo = df_orig[df_orig.PUNTO_MEDICION==punto].PORCENTAJE_PACK_LOSS.describ
e()[-1]
    desv = df_orig[df_orig.PUNTO_MEDICION==punto].PORCENTAJE_PACK_LOSS.describe
()[2]
    if media!=0:
        pass
        print("{:60}".format(f'La media en {punto} es ')+ f'{media} , maximo \t
{maximo}y desviacion \t{desv}')
```

```
La media en ABA - Temple es                                  0.2543154994066051
4 , maximo      1.6521221060236622y desviacion  0.42606524858491324
La media en BAZ - Carlos Paz es                              0.1567515755019720
2 , maximo      0.8249194807842212y desviacion  0.24321130334201563
La media en BAZ - Yocsina es                                 0.1508550858732157
5 , maximo      5.10260003065611y desviacion    0.36634137747320544
La media en Carlos Paz - Cosquin es                          0.2106958897156687
2 , maximo      3.0450946942347903y desviacion  0.2819936041628785
La media en Carlos Paz - La Falda es                         0.1679500396071416
, maximo        3.1080020389594303y desviacion  0.2661909602122908
La media en EDC - Capitalinas es                             0.1097672350766518
3 , maximo      2.2145713441161803y desviacion  0.15135509939062766
La media en EDC - Coral State es                             0.6360028046877939
, maximo        7.92859482335571y desviacion    0.9454587210203067
La media en EDC - ET Oeste es                                0.1477209952412642
, maximo        2.1382085289290904y desviacion  0.25026106666789605
La media en EDC - MOP es                                     0.0964723138243349
5 , maximo      3.35744486697923y desviacion    0.17108208069772227
La media en EDC - RDB es                                     0.0724590580809786
, maximo        2.19263093656621y desviacion    0.1439026972804306
La media en JM - Totoral Vieja es                            0.1473471648760014
7 , maximo      2.5082909398065y desviacion     0.24231350286725115
La media en NOC - 6720HI to BAZ es                           0.0878087227656291
9 , maximo      0.4142508471207163y desviacion  0.12721632217709772
La media en NOC - 6720HI to ETC es                           0.1937625225594512
8 , maximo      0.8274471937942701y desviacion  0.255506232229156
La media en NOC - 6720HI to RPrivado es                      0.07377825625146 ,
maximo  0.4170796799472919y desviacion  0.1054713035448317
La media en NOC - ACHALA - Servicios es                      1.0211577902712916
, maximo        58.28972430108815y desviacion   6.57923845565565
La media en NOC - ACHALA - Solo Dolores es                   3.596557140315091
, maximo        54.54599590895909y desviacion   7.181847599225291
La media en NOC - Almacenes es                               0.1112287164743989
7 , maximo      0.5827555239400188y desviacion  0.1712271550866733
La media en NOC - ET Sur es                                  0.2053192902962070
5 , maximo      0.7910361534507657y desviacion  0.24941277129414757
La media en NOC - Interfabricas es                           0.0508033794580440
4 , maximo      0.2470621920754761y desviacion  0.07379373515150434
La media en NOC - S9306 to SS6720HI es                       0.6032602383053347
, maximo        2.6499238868537014y desviacion  0.8171475591203824
La media en NOC - SW Clientes 1 es                           0.0576571533611105
54 , maximo     0.3193328874230708y desviacion  0.09108530681434703
La media en NOC - SW Clientes 2 es                           0.0668393442842613
5 , maximo      0.3172032002068044y desviacion  0.09561918032474216
La media en RDB - Escuela de Capacitacion es                 0.0941362417302148
1 , maximo      0.4185183860494317y desviacion  0.12935377876769213
La media en RDB - ET La Calera es                            0.1607584608947408
8 , maximo      0.7803917815266541y desviacion  0.21129378484860906
La media en RDB - JM es                                      0.1054092011943265
6 , maximo      0.4182166631127375y desviacion  0.1319538041217017
La media en RDB - PEA es                                     0.5314621751667343
, maximo        8.25970242902841y desviacion    1.3734194250366634
La media en RDB - RIO es                                     0.2690503301923664
, maximo        50.6259679769207y desviacion    1.4282973080094041
La media en SF - Freyre es                                   0.2538765833450658
, maximo        1.2307724456010172y desviacion  0.3569860885523234
La media en SF - La Francia es                               0.1782176297228803
, maximo        1.2324320360885728y desviacion  0.3067148020569498
La media en SF - Las Varillas es                             0.1969099958088807
7 , maximo      1.2203922233770412y desviacion  0.29766105369003354
La media en SF - SF Adm es                                   0.2523190861487769
, maximo        1.1459081176024244y desviacion  0.3194595193280621
La media en SF - SF Cliente es                               0.2180890334985812
5 , maximo      1.1804027461868225y desviacion  0.3143680475919922
La media en Yocsina - Alta Gracia es                         0.0491229284961966
1 , maximo      0.20924860791205316y desviacion         0.06497431108625883
La media en Yocsina - Carlos Paz es                          0.0560626903228591
9 , maximo      0.20996400179596364y desviacion         0.0656116418469502
```

```
La media en Yocsina - Mogote es                                0.0341132277005533
34 , maximo     0.42701439853982204y desviacion         0.05878221924236231
```

Elegimos 'NOC - ACHALA - Servicios' debido a que tiene una feature con buena media y una dispercion en sus valores acotada.

In [25]:
```
df = df_orig.dropna(subset=['LATENCIA_MS', 'OUTBOUND_BITS', 'INBOUND_BITS', 'POR
CENTAJE_PACK_LOSS'])
```

In [26]:
```
# Separamos el "target" del resto del dataset
X = df.loc[df.PUNTO_MEDICION == 'NOC - ACHALA - Servicios', :]
y = X['PORCENTAJE_PACK_LOSS']
X = X.loc[:, X.columns != 'PORCENTAJE_PACK_LOSS']
```

In [27]:
```
labels=[0, 1, 2]
y_cat = pd.cut(y, [0,0.05,0.1,np.inf], labels=labels, right=False)
```

In [28]:
```
y_cat.unique()
```

Out[28]:
```
[2, 1, 0]
Categories (3, int64): [0 < 1 < 2]
```

Seleccionamos uno o más feature del dataset que no sea categórico, por ejemplo `INBOUND_BITS` y `OUTBOUND_BITS`

In [29]:
```
# TODO: modificar esta feature por algún otro (o una combinacion de estos) para v
er como cambian los resultados
#X = X[['INBOUND_BITS', 'OUTBOUND_BITS']]
#X = df.loc[:, df.columns != 'PORCENTAJE_PACK_LOSS']
X = X.loc[:, X.columns != 'PUNTO_MEDICION']
X = X.loc[:, X.columns != 'FECHA_INICIO_MEDICION']
X = X.loc[:, X.columns != 'FECHA_FIN_MEDICION']
X = X.loc[:, X.columns != 'FECHA_HORA']
X = X.loc[:, X.columns != 'MEDIDA']
X = X.loc[:, X.columns != 'ID_EQUIPO']
```

In [30]:
```
X = X.loc[:, X.columns != 'CAPACIDAD_MAXIMA_GBS']
X = X.loc[:, X.columns != 'PASO']
X = X.loc[:, X.columns != 'LATENCIA_MS']
X.columns
```

Out[30]:
```
Index(['INBOUND_BITS', 'OUTBOUND_BITS'], dtype='object')
```

División de datos en conjuntos de entrenamiento y evaluación

La primer tarea consiste en dividir el conjunto de datos cargados en el apartado anterior en conjuntos de entrenamiento (training) y evaluación (test).

Utilizar aproximadamente 70% de los datos para entrenamiento y 30% para validación.

Links:

- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)

In [31]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y_cat, test_size=0.3, rand
om_state=77)
```

Regresion Logistica

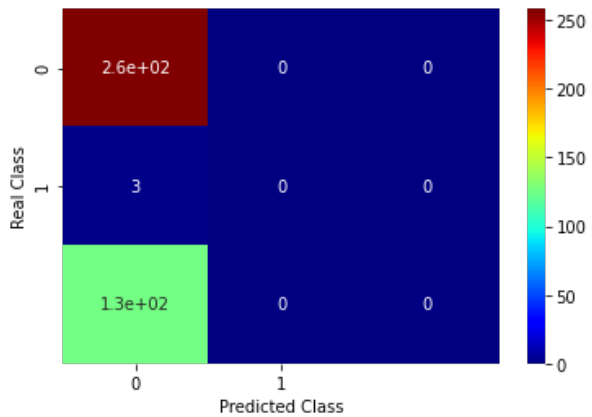
Link:

- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)

In [32]:
```
def print_classification_report(y_real, y_pred):
    print(f"Accuracy {accuracy_score(y_real, y_pred)}")
    print("-"*100)
    print(classification_report(y_real, y_pred, zero_division=0))
    sns.heatmap(confusion_matrix(y_real, y_pred),
                xticklabels=[0, 1],
                yticklabels=[0, 1],
                cmap="jet",
                annot=True,
               )
    plt.xlabel("Predicted Class")
    plt.ylabel("Real Class")
```

In [33]:
```
penalty = 'l2'# TODO: Tipo de regularización: l1 (valor absoluto), l2 (cuadrado
s).
alpha = 0.00001# TODO: Parámetro de regularización. También denominado como pará
metro `lambda`. Debe ser mayor que 0.

scaler = preprocessing.StandardScaler()
model = LogisticRegression(penalty=penalty, C=1./alpha, multi_class='ovr')
pipe = Pipeline(steps=[("scaler", scaler), ("logistic_reg", model)])
pipe = model
pipe.fit(X_train, y_train)

print(f"Accuracy para entrenamiento: {accuracy_score(y_train, pipe.predict(X_tra
in)):.2f}")
print(f"Accuracy para validación   : {accuracy_score(y_test, pipe.predict(X_tes
t)):.2f}")

print(f"MSE para entrenamiento: {mean_squared_error(y_train, pipe.predict(X_trai
n)):.2f}")
print(f"MSE para validación   : {mean_squared_error(y_test, pipe.predict(X_tes
t)):.2f}")

print_classification_report(y_test, pipe.predict(X_test))
```

```
Accuracy para entrenamiento: 0.64
Accuracy para validación   : 0.67
MSE para entrenamiento: 1.42
MSE para validación   : 1.31
Accuracy 0.6666666666666666
---------------------------------------------------------------------------
----------------------
              precision    recall  f1-score   support

           0       0.67      1.00      0.80       258
           1       0.00      0.00      0.00         3
           2       0.00      0.00      0.00       126

    accuracy                           0.67       387
   macro avg       0.22      0.33      0.27       387
weighted avg       0.44      0.67      0.53       387
```



Esta es la matriz de confusion para el conjunto de entrenamiento, ahora la graficaremos usando el conjunto de validacion.

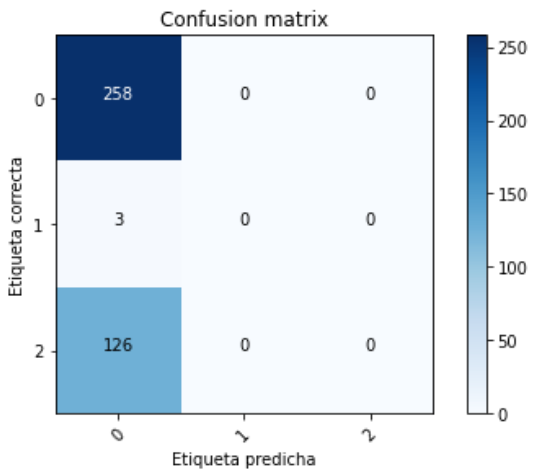
Matriz de Confusion

Plotear las matrices de confunsion y sacar conclusiones

In [34]:
```
import itertools
```

In [35]:
```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    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")

    plt.tight_layout()
    plt.ylabel('Etiqueta correcta')
    plt.xlabel('Etiqueta predicha')
```

In [36]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, pipe.predict(X_test))
#tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
plot_confusion_matrix(cm, [0, 1, 2])
```



Despues de probar con muchas combinaciones de features, encontramos que el modelo de regresion lineal que mejor accuracy otorga, es aquel que categoriza todo bajo la misma label. Esto creemos que se debe en parte a que los datos estan desbalanceados (no hay la misma cantidad de datos para cada categoria).

Analisis Extra

Graficos para determinar por inspeccion las posibles correlaciones entre las features

In [37]:
```
df_filtered = df.loc[df.PUNTO_MEDICION == 'RDB - RIO', :]
```

In [39]:
```
feature_names = np.array(df_filtered.columns)
```

In [40]:
```
objetivo = 'LATENCIA_MS'
feature_names
```

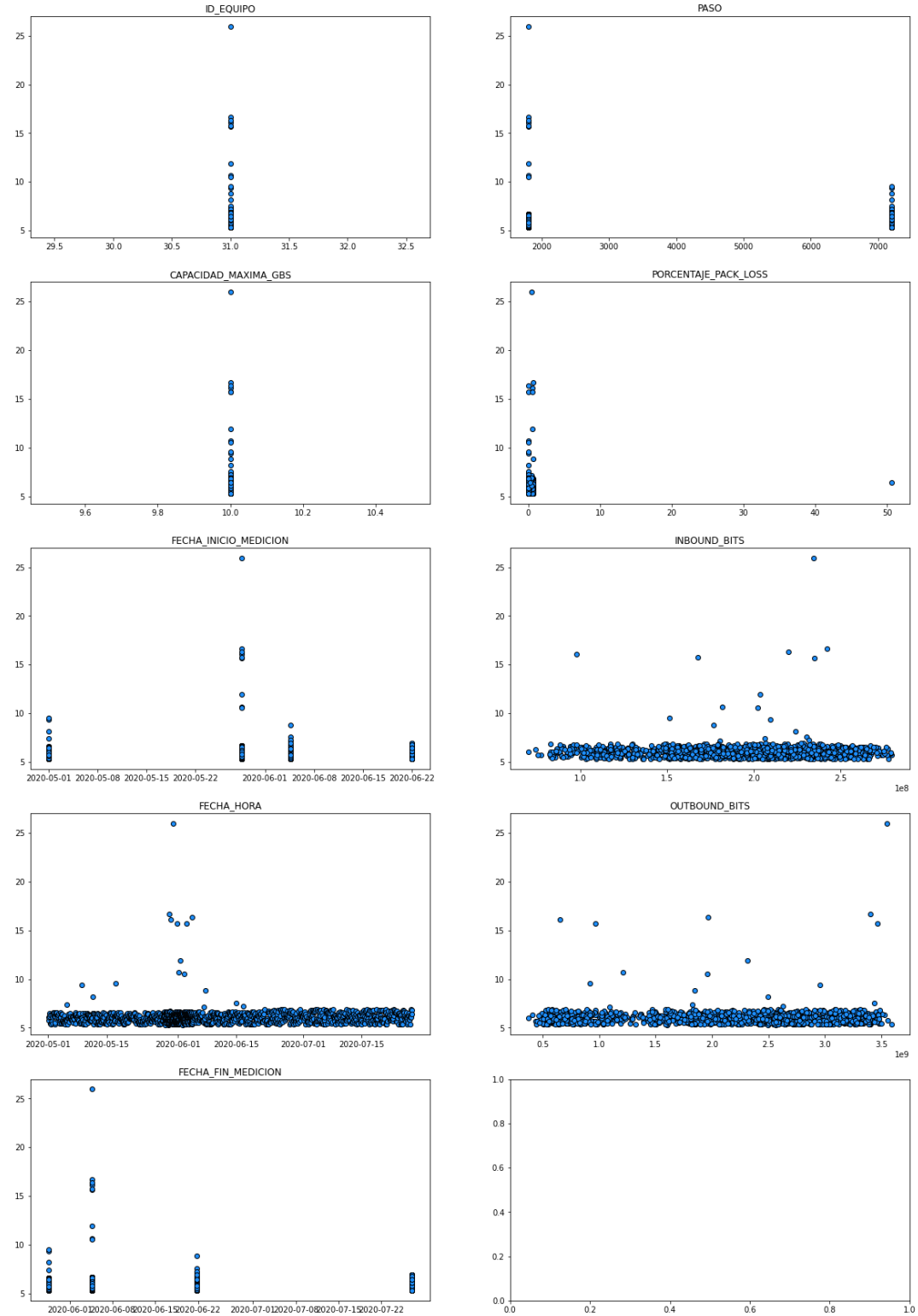
Out[40]:
```
array(['ID_EQUIPO', 'PUNTO_MEDICION', 'CAPACIDAD_MAXIMA_GBS',
       'FECHA_INICIO_MEDICION', 'FECHA_HORA', 'FECHA_FIN_MEDICION',
       'PASO', 'LATENCIA_MS', 'PORCENTAJE_PACK_LOSS', 'INBOUND_BITS',
       'OUTBOUND_BITS', 'MEDIDA'], dtype=object)
```

In [56]:
```
feature_names = np.delete(feature_names, np.where(feature_names==objetivo))
feature_names = np.delete(feature_names, np.where(feature_names=='PUNTO_MEDICION
'))
feature_names = np.delete(feature_names, np.where(feature_names=='MEDIDA'))
```

In [57]:
```
feature_names
```

Out[57]:
```
array(['ID_EQUIPO', 'CAPACIDAD_MAXIMA_GBS', 'FECHA_INICIO_MEDICION',
       'FECHA_HORA', 'FECHA_FIN_MEDICION', 'PASO', 'PORCENTAJE_PACK_LOSS',
       'INBOUND_BITS', 'OUTBOUND_BITS'], dtype=object)
```

In [58]:
```
from matplotlib import pyplot as plt
fig, ax = plt.subplots(5,2, figsize=(20,30))
y = df_filtered[objetivo]

for indice, feat in enumerate(feature_names):
  ax[indice%5][indice//5].scatter(df_filtered[feat], y, facecolor="dodgerblue", e
dgecolor="k", label="datos")
  ax[indice%5][indice//5].set_title(feat)
  #ax[indice%6][indice//6].set_ylim(0,1)
plt.show()
```



Ahora realizamos el mismo analisis pero con la variable PORCENTAJE_PACK_LOSS como objetivo.

In [59]:
```
objetivo = 'PORCENTAJE_PACK_LOSS'
feature_names
```

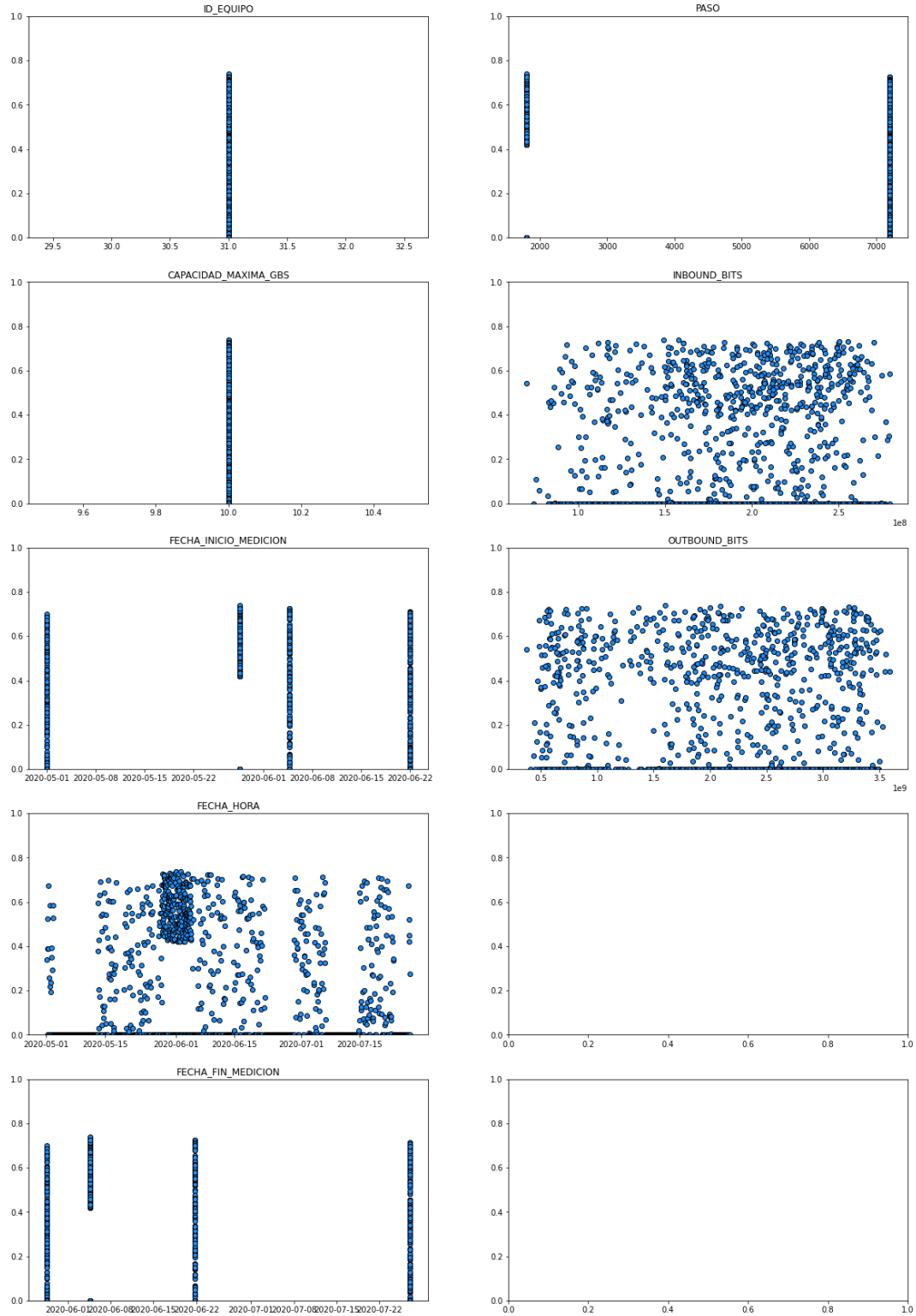
Out[59]:
```
array(['ID_EQUIPO', 'CAPACIDAD_MAXIMA_GBS', 'FECHA_INICIO_MEDICION',
       'FECHA_HORA', 'FECHA_FIN_MEDICION', 'PASO', 'PORCENTAJE_PACK_LOSS',
       'INBOUND_BITS', 'OUTBOUND_BITS'], dtype=object)
```

In [60]:
```
feature_names = np.delete(feature_names, np.where(feature_names==objetivo))
feature_names = np.delete(feature_names, np.where(feature_names=='PUNTO_MEDICION
'))
feature_names = np.delete(feature_names, np.where(feature_names=='MEDIDA'))
```

In [61]:
```
feature_names
```

Out[61]:
```
array(['ID_EQUIPO', 'CAPACIDAD_MAXIMA_GBS', 'FECHA_INICIO_MEDICION',
       'FECHA_HORA', 'FECHA_FIN_MEDICION', 'PASO', 'INBOUND_BITS',
       'OUTBOUND_BITS'], dtype=object)
```

In [64]:
```
from matplotlib import pyplot as plt
fig, ax = plt.subplots(5,2, figsize=(20,30))
y = df_filtered[objetivo]

for indice, feat in enumerate(feature_names):
  ax[indice%5][indice//5].scatter(df_filtered[feat], y, facecolor="dodgerblue", e
dgecolor="k", label="datos")
  ax[indice%5][indice//5].set_title(feat)
  ax[indice%5][indice//5].set_ylim(0,1)
plt.show()
```



Notar en ambas graficas (latencia y packet loss) como en la fechahora alrededor del primero de junio cambia mucho la distibucion de las muestras. Por ahora no podemos deducir alguna causa, pero es bueno notarlo.

Aparte de eso, no podemos ver ninguna correlacion significativa.

Ahora probaremos con otros modelos, haciendo uso del Grid para alternar entre diferentes configuraciones de los mismos

Empezamos con un arbol de desicion

In [43]:
```
tree_clf = DecisionTreeClassifier(random_state=77)
```

In [65]:
```
param_grid = {
    "criterion":['gini', 'entropy'],
    "max_depth": [2,4,5, 7, 10, 12],
    "min_samples_leaf": [1, 2, 3, 5, 7, 10, 15, 20],
    }
grid_tree = GridSearchCV(tree_clf, param_grid=param_grid, cv=5, scoring="accurac
y")
```

In [66]:
```
grid_tree.fit(X_train, y_train)
```

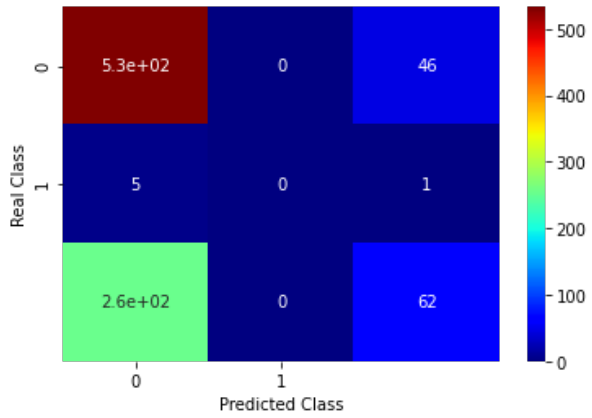
Out[66]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=77),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [2, 4, 5, 7, 10, 12],
                         'min_samples_leaf': [1, 2, 3, 5, 7, 10, 15, 20]},
             scoring='accuracy')
```

In [67]:
```
means = grid_tree.cv_results_['mean_test_score']
stds = grid_tree.cv_results_['std_test_score']
top_5 = 0
for mean, std, params in sorted(zip(means, stds, grid_tree.cv_results_['params
']),
                                key=lambda data: data[0], reverse=True):
    print("Mean accuracy = %0.4f (+/-%0.04f) para %r" % (mean, std * 2, params))
    top_5 += 1
    if top_5 == 5: break
print()
```

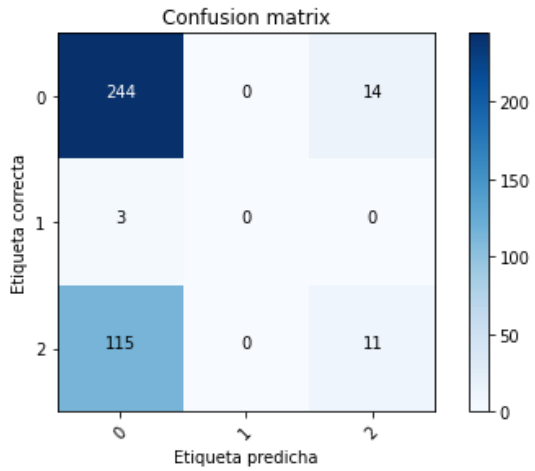
```
Mean accuracy = 0.6412 (+/-0.0361) para {'criterion': 'gini', 'max_depth': 4,
'min_samples_leaf': 15}
Mean accuracy = 0.6412 (+/-0.0248) para {'criterion': 'entropy', 'max_depth':
2, 'min_samples_leaf': 7}
Mean accuracy = 0.6401 (+/-0.0288) para {'criterion': 'gini', 'max_depth': 2,
'min_samples_leaf': 15}
Mean accuracy = 0.6379 (+/-0.0372) para {'criterion': 'entropy', 'max_depth':
2, 'min_samples_leaf': 15}
Mean accuracy = 0.6379 (+/-0.0214) para {'criterion': 'entropy', 'max_depth':
2, 'min_samples_leaf': 1}
```

In [68]:
```
best_tree = grid_tree.best_estimator_
best_tree.fit(X_train, y_train)
y_pred_train = best_tree.predict(X_train)
print_classification_report(y_train, y_pred_train)
```

```
Accuracy 0.6589147286821705
-------------------------------------------------------------------------------
----------------------
              precision    recall  f1-score   support

           0       0.67      0.92      0.78       579
           1       0.00      0.00      0.00         6
           2       0.57      0.19      0.29       318

    accuracy                           0.66       903
   macro avg       0.41      0.37      0.36       903
weighted avg       0.63      0.66      0.60       903
```



In [48]:
```
cm = confusion_matrix(y_test, best_tree.predict(X_test))
#tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
plot_confusion_matrix(cm, [0, 1, 2])
```



Vemos que el arbol de desiciones lo hace un poco mejor, al considerar otras categorias a parte de la predominante, pero sin embargo sigue teniendo errores considerables en la clasificacion.

Un problema que detectamos es que la clasificacion de la categoria del medio ([0.05,0.1)⇒ 1) no esta bien representada por la distribucion de muestras, es decir, no hay suficientes mediciones que esten dentro de esta categoria, por lo que el modelo no puede aprender a discriminarlas.

Una solucion seria mover las barreras que delimitan las categorias.