**DiploDatos 2019 - Análisis de Series Temporales**[**¶**](#gjdgxs)

## **Integrante**[**¶**](#30j0zll)

| **Nombre** | **e-mail** |
| --- | --- |
| Rivadero, Isabel | isarivadero@hotmail.com |

## **Práctico de Aprendizaje supervisado y no supervisado**[**¶**](#1fob9te)

Continuo trabajando sobre aprendizaje automático: sobre modelos supervisados y no supervisados. Diseño e implemento algunos modelos y defino métricas para ver como performan.

In [1]:

*### Aumentar el ancho del notebook*  
**from** **IPython.core.display** **import** display, HTML  
display(HTML("<style>.container { width:90% !important; }</style>"))

In [2]:

%**matplotlib** inline  
**import** **numpy** **as** **np**  
**import** **pandas** **as** **pd**  
**import** **seaborn** **as** **sns**  
**import** **matplotlib.pyplot** **as** **plt**  
**from** **sklearn.model\_selection** **import** train\_test\_split  
**from** **sklearn.pipeline** **import** Pipeline  
**from** **sklearn.metrics** **import** accuracy\_score, precision\_score, recall\_score, f1\_score  
**from** **sklearn.preprocessing** **import** MinMaxScaler  
  
plt.rcParams['figure.figsize'] = (12,8)  
  
holidays = './holidays.csv'  
cols = ['service',  
 'sender\_zipcode',  
 'receiver\_zipcode',  
 'sender\_state',  
 'receiver\_state',  
 'shipment\_type',  
 'quantity',  
 'status',  
 'date\_created',  
 'date\_sent',  
 'date\_visit',  
 'target']  
data\_path = './shipments\_BR\_201903.csv'

In [3]:

**def** ontimesv(y\_test, y\_pred):  
 ontime\_sv= (y\_pred == y\_test)  
 **return** np.sum(ontime\_sv) / np.size(y\_test)  
  
**def** delaysv(y\_test, y\_pred):  
 delay\_sv = (y\_pred < y\_test)  
 **return** np.sum(delay\_sv) / np.size(y\_test)  
  
**def** earlysv(y\_test, y\_pred):  
 early\_sv= (y\_test < y\_pred)  
 **return** np.sum(early\_sv) / np.size(y\_test)  
  
**def** offset\_window(y\_test, lower\_bound, upper\_bound, length):  
 offset\_msk = ((upper\_bound - lower\_bound) == length)  
 **return** np.sum(offset\_msk) / np.size(offset\_msk)  
  
  
**def** avg\_speed(y\_test, lower\_bound, upper\_bound):  
 **return** lower\_bound.mean()  
  
  
**def** avg\_offset(y\_test, lower\_bound, upper\_bound):  
 **return** (upper\_bound - lower\_bound).mean()  
  
**def** get\_metrics(y\_test, speed, offset):  
 lower\_bound = speed  
 upper\_bound = speed + offset  
 metrics = {'on\_time': ontime(y\_test, lower\_bound, upper\_bound).astype(float).round(3),  
 'delay': delay(y\_test, lower\_bound, upper\_bound).astype(float).round(3),  
 'early': early(y\_test, lower\_bound, upper\_bound).astype(float).round(3),  
 'offset\_0': offset\_window(y\_test, lower\_bound, upper\_bound, 0).astype(float).round(3),  
 'offset\_1': offset\_window(y\_test, lower\_bound, upper\_bound, 1).astype(float).round(3),  
 'offset\_2': offset\_window(y\_test, lower\_bound, upper\_bound, 2).astype(float).round(3),  
 'avg\_speed': avg\_speed(y\_test, lower\_bound, upper\_bound).astype(float).round(3),  
 'avg\_offset': avg\_offset(y\_test, lower\_bound, upper\_bound).astype(float).round(3),  
 }  
  
 **return** metrics

**Referencia de las columnas**[**¶**](#3znysh7)

* **service**: Identificador unico que corresponde a un tipo de servicio de un correo en particular.
* **sender\_zipcode:** Código postal de quien envía el paquete (usualmente el vendedor).
* **receiver\_zipcode:** Código postal de quien recibe el paquete (usualmente el comprador).
* **sender\_state:** Nombre abreviado del estado de quien envía el paquete.
* **receiver\_state:** Nombre abreviado del estado de quien recibe el paquete.
* **quantity:** Cantidad de items que tiene dentro el paquete.
* **status:** Estado final del envío.
* **date\_created:** Fecha de compra de el o los items.
* **date\_sent:** Fecha en que el correo recibe el paquete.
* **date\_visit:** Fecha en que el correo entrega el paquete.
* **target:** Cantidad de dias hábiles que tardó el correo en entregar el paquete desde que lo recibe.

In [4]:

df = pd.read\_csv(data\_path, usecols=cols)  
df.shape

Out[4]:

(1000000, 12)

In [5]:

df.head()

Out[5]:

|  | **sender\_state** | **sender\_zipcode** | **receiver\_state** | **receiver\_zipcode** | **shipment\_type** | **quantity** | **service** | **status** | **date\_created** | **date\_sent** | **date\_visit** | **target** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | SP | 3005 | SP | 5409 | express | 1 | 0 | done | 2019-03-04 00:00:00 | 2019-03-05 13:24:00 | 2019-03-07 18:01:00 | 2 |
| **1** | SP | 17052 | MG | 37750 | standard | 1 | 1 | done | 2019-03-19 00:00:00 | 2019-03-20 14:44:00 | 2019-03-27 10:21:00 | 5 |
| **2** | SP | 2033 | SP | 11040 | express | 1 | 0 | done | 2019-02-18 00:00:00 | 2019-02-21 15:08:00 | 2019-02-28 18:19:00 | 5 |
| **3** | SP | 13900 | SP | 18500 | express | 1 | 0 | done | 2019-03-09 00:00:00 | 2019-03-11 15:48:00 | 2019-03-12 13:33:00 | 1 |
| **4** | SP | 4361 | RS | 96810 | express | 1 | 0 | done | 2019-03-08 00:00:00 | 2019-03-12 08:19:00 | 2019-03-16 08:24:00 | 4 |

In [6]:

*# set seed for reproducibility*  
np.random.seed(0)

In [7]:

df.dtypes

Out[7]:

sender\_state object  
sender\_zipcode int64  
receiver\_state object  
receiver\_zipcode int64  
shipment\_type object  
quantity int64  
service int64  
status object  
date\_created object  
date\_sent object  
date\_visit object  
target int64  
dtype: object

Como las fechas estan como tipo objeto no voy a poder operar entonces paso a tipo fecha

In [8]:

df1= df.copy()

In [9]:

df1['date\_created']= df1.date\_created.astype('datetime64')

In [10]:

df1['date\_sent']= df1.date\_sent.astype('datetime64')

In [11]:

df1['date\_visit']= df1.date\_visit.astype('datetime64')

**Eliminamos datos inconsistentes:**

Aplicamos curacion y limpieza de datos

In [12]:

df1 = df1[(df1['date\_sent'] <= df1['date\_visit']) & (df1['date\_created'] <= df1['date\_sent']) & (df1['date\_created'] <= df1['date\_visit'])]  
df1.shape

Out[12]:

(999827, 12)

**Preparación de los features**[**¶**](#2et92p0)

#### **Diseñar un pipeline con las siguientes transformaciones:**[**¶**](#tyjcwt)

1- Recortar el último dígito de los zip codes

In [13]:

df1['sender\_zipcode']=df1['sender\_zipcode']/10  
df1['sender\_zipcode'] =df1['sender\_zipcode'].astype('int32')  
df1['receiver\_zipcode']=df1['receiver\_zipcode']/10  
df1['receiver\_zipcode'] =df1['receiver\_zipcode'].astype('int32')

In [14]:

df1.dtypes

Out[14]:

sender\_state object  
sender\_zipcode int32  
receiver\_state object  
receiver\_zipcode int32  
shipment\_type object  
quantity int64  
service int64  
status object  
date\_created datetime64[ns]  
date\_sent datetime64[ns]  
date\_visit datetime64[ns]  
target int64  
dtype: object

In [15]:

df1.head()

Out[15]:

|  | **sender\_state** | **sender\_zipcode** | **receiver\_state** | **receiver\_zipcode** | **shipment\_type** | **quantity** | **service** | **status** | **date\_created** | **date\_sent** | **date\_visit** | **target** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | SP | 300 | SP | 540 | express | 1 | 0 | done | 2019-03-04 | 2019-03-05 13:24:00 | 2019-03-07 18:01:00 | 2 |
| **1** | SP | 1705 | MG | 3775 | standard | 1 | 1 | done | 2019-03-19 | 2019-03-20 14:44:00 | 2019-03-27 10:21:00 | 5 |
| **2** | SP | 203 | SP | 1104 | express | 1 | 0 | done | 2019-02-18 | 2019-02-21 15:08:00 | 2019-02-28 18:19:00 | 5 |
| **3** | SP | 1390 | SP | 1850 | express | 1 | 0 | done | 2019-03-09 | 2019-03-11 15:48:00 | 2019-03-12 13:33:00 | 1 |
| **4** | SP | 436 | RS | 9681 | express | 1 | 0 | done | 2019-03-08 | 2019-03-12 08:19:00 | 2019-03-16 08:24:00 | 4 |

2- Normalizar los features para que queden en el rango (0, 1)

In [16]:

features = ['sender\_zipcode', 'receiver\_zipcode', 'service']  
target = 'target'

In [17]:

df1\_x=df[features].copy()  
df1\_y=df[target].copy()

In [18]:

scaler = MinMaxScaler()  
scaler.fit(df1\_x)  
df1\_xn=scaler.transform(df1\_x)

/home/isanti/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/data.py:323: DataConversionWarning: Data with input dtype int64 were all converted to float64 by MinMaxScaler.  
 return self.partial\_fit(X, y)

In [19]:

scaler = MinMaxScaler()  
scaler.fit([df1\_y])  
df1\_yn=scaler.transform([df1\_y])

3- Proyectar los features utilizando PCA, manteniendo 3 componentes.

#### **NOTA IMPORTANTE:**[**¶**](#3dy6vkm)

Estas transformaciones se deben aplicar sin modificar el dataframe con los datos originales, pueden usar copias para hacer las pruebas. Es decir que no deben hacer las transformaciones y guardarlas en un dataframe, tal como se hace en el ejemplo.

In [20]:

**from** **sklearn.decomposition** **import** PCA

In [21]:

pca = PCA(n\_components=3)  
pca.fit(df1\_xn)

Out[21]:

PCA(copy=True, iterated\_power='auto', n\_components=3, random\_state=None,  
 svd\_solver='auto', tol=0.0, whiten=False)

In [22]:

pca.n\_components\_

Out[22]:

3

In [23]:

pca.transform(df1\_xn)

Out[23]:

array([[-0.41161583, -0.03821374, -0.11852587],  
 [-0.05729793, -0.03615093, -0.02055959],  
 [-0.3625151 , -0.06853404, -0.1199857 ],  
 ...,  
 [-0.2835178 , 0.30530984, -0.08923477],  
 [-0.35669799, 0.03232526, 0.2528347 ],  
 [-0.37762851, 0.09594395, -0.01618233]])

**Preparación del target:**[**¶**](#1t3h5sf)

4- Limitar el target a 20, es decir asignar todo target mayor que 20 a 20.

In [24]:

df1=df.copy()  
df1.loc[df1['target']>20, 'target'] = 20

In [25]:

df1['sender\_zipcode']=df1['sender\_zipcode']/10  
df1['sender\_zipcode'] =df1['sender\_zipcode'].astype('int32')  
df1['receiver\_zipcode']=df1['receiver\_zipcode']/10  
df1['receiver\_zipcode'] =df1['receiver\_zipcode'].astype('int32')

In [26]:

df1['date\_created']= df1.date\_created.astype('datetime64')  
df1['date\_sent']= df1.date\_sent.astype('datetime64')  
df1['date\_visit']= df1.date\_visit.astype('datetime64')

In [27]:

df1 = df1[(df1['date\_sent'] <= df1['date\_visit']) & (df1['date\_created'] <= df1['date\_sent']) & (df1['date\_created'] <= df1['date\_visit'])]  
df1.shape

Out[27]:

(999827, 12)

In [28]:

df1.target.value\_counts()

Out[28]:

1 159537  
2 135145  
3 107500  
4 82250  
5 68790  
6 59190  
7 51902  
8 47937  
9 42376  
10 40461  
11 35821  
12 30215  
13 25068  
20 21338  
0 21053  
14 19853  
15 16241  
16 12133  
17 9623  
18 7339  
19 6055  
Name: target, dtype: int64

**Preparación del dataset:**[**¶**](#4d34og8)

5- Particionar el dataset en train y test, teniendo los cuidados necesarios para no romper la temporalidad de los datos. El conjunto de training no puede tener menos del 50% de los datos.

In [29]:

cut\_off = '2019-03-20'  
df\_train = df1.query(f'date\_visit <= "**{cut\_off}**"')  
df\_test = df1.query(f'date\_created > "**{cut\_off}**"')  
  
X\_train = df\_train[features].values.astype(np.float)  
y\_train = df\_train[target].values  
  
X\_test = df\_test[features].values.astype(np.float)  
y\_test = df\_test[target].values  
  
X\_train.shape, y\_train.shape, X\_test.shape, y\_test.shape

Out[29]:

((673474, 3), (673474,), (56694, 3), (56694,))

6- Si les parece necesario, pueden realizar algún tipo de filtrado o limpieza de los datos, explicando por qué les parece necesario.

ya filtre los datos con fechas inconsistentes anteriormente

**Modelo basado en árboles de decisión (supervisado)**[**¶**](#2s8eyo1)

7- Crear un pipeline con los pasos de “preparación de los features” agregando el clasificador XGBoostClassifier como estimador final. Entrenar este modelo, predecir el conjunto de test y calcular las métricas ontime, delay y early, sin ventana (se puede utilizar un array con ceros como en el ejemplo).

In [30]:

**from** **xgboost** **import** XGBClassifier

In [31]:

model = Pipeline([  
 ('normalizer', MinMaxScaler()),  
 ('PCA',PCA(n\_components=3)),  
 ('classifier', XGBClassifier(disable\_default\_eval\_metric=1,nthread=5)),  
])

In [34]:

%%time  
model.fit(X\_train, y\_train)

CPU times: user 37min 31s, sys: 13min 1s, total: 50min 32s  
Wall time: 13min 2s

Out[34]:

Pipeline(memory=None,  
 steps=[('normalizer', MinMaxScaler(copy=True, feature\_range=(0, 1))), ('PCA', PCA(copy=True, iterated\_power='auto', n\_components=3, random\_state=None,  
 svd\_solver='auto', tol=0.0, whiten=False)), ('classifier', XGBClassifier(base\_score=0.5, booster='gbtree', colsample\_bylevel=1,  
 colsample\_by...lpha=0, reg\_lambda=1, scale\_pos\_weight=1,  
 seed=None, silent=None, subsample=1, verbosity=1))])

In [35]:

y\_pred = model.predict(X\_test)

In [36]:

y\_pred

Out[36]:

array([5, 7, 2, ..., 2, 1, 1])

In [37]:

metrics = {  
 'ontimesv': ontimesv(y\_test, y\_pred),  
 'delaysv': delaysv(y\_test, y\_pred),  
 'earlysv': earlysv(y\_test, y\_pred),  
}  
  
metrics

Out[37]:

{'ontimesv': 0.5394927152785127,  
 'delaysv': 0.22330405333897768,  
 'earlysv': 0.23720323138250962}

El 22 por ciento llego con retaso, el 24 por cierto antes y el 54 por ciento en tiempo. Mas del 75 por ciento llego en tiempo o antes de lo predicho.

8- Explicar muy brevemente como funcionan esta clase de modelos.

XGBoostClassifier se trata de arboles de decision con gradient boosting (que es una técnica para regresión y clasificación dónde se minimizan los errores por gradient descendent y que produce un modelo tipo árbol de decisión). Es gradient boosting mejorado ya que es una optimización.

**Modelo basado en vecinos cercanos (no supervisado / semi supervisado)**[**¶**](#17dp8vu)

9- Crear un pipeline con los pasos de “preparación de los features” agregando el clasificador KNeighborsClassifier como estimador final. Entrenar este modelo, predecir el conjunto de test y calcular las métricas ontime, delay y early, sin ventana.

In [38]:

**from** **sklearn.neighbors** **import** KNeighborsClassifier

In [39]:

model = Pipeline([  
 ('normalizer', MinMaxScaler()),  
 ('PCA',PCA(n\_components=3)),  
 ('classifier', KNeighborsClassifier(n\_neighbors=14)),  
])

In [40]:

%%time  
model.fit(X\_train, y\_train)

CPU times: user 2.73 s, sys: 600 ms, total: 3.33 s  
Wall time: 4.15 s

Out[40]:

Pipeline(memory=None,  
 steps=[('normalizer', MinMaxScaler(copy=True, feature\_range=(0, 1))), ('PCA', PCA(copy=True, iterated\_power='auto', n\_components=3, random\_state=None,  
 svd\_solver='auto', tol=0.0, whiten=False)), ('classifier', KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski',  
 metric\_params=None, n\_jobs=None, n\_neighbors=14, p=2,  
 weights='uniform'))])

In [41]:

y\_pred = model.predict(X\_test)

In [42]:

y\_pred

Out[42]:

array([ 5, 15, 1, ..., 1, 1, 2])

In [43]:

metrics = {  
 'ontimesv': ontimesv(y\_test, y\_pred),  
 'delaysv': delaysv(y\_test, y\_pred),  
 'earlysv': earlysv(y\_test, y\_pred),  
}  
  
metrics

Out[43]:

{'ontimesv': 0.5231065015698311,  
 'delaysv': 0.21339118778001198,  
 'earlysv': 0.263502310650157}

El 21 por ciento de las predicciones llego mas tarde de lo que se predijo, el 26 por ciento de las predicciones llegaron mas temprano de lo predicho mientras que el 52 por ciento llego en el tiempo predicho. Podriamos decir que el modelo anda muy bien porque mas del 75 por ciento llego antes o en el tiempo predicho.

10- Explicar muy brevemente como funcionan esta clase de modelos.

KNN se trata de un algoritmo de clasificacion de aprendizaje supervisado que toma puntos etiquetados y los usa para aprender como etiquetar otros puntos. Clasifica casos basados en su similitud con otros casos usando que casos similares con las mismas etiquetas de clase estan cerca una de otras(Los datos que estan cerca son llamados "vecinos") por lo tanto la distancia es una medida de similitud.

Modelo basado en regresión: 11- Crear un pipeline con los pasos de “preparación de los features” agregando un regresor a elección de ustedes como estimador final. Este regresor puede ser tanto supervisado como no supervisado.

In [44]:

**from** **sklearn.linear\_model** **import** LogisticRegression

In [45]:

model = Pipeline([  
 ('normalizer', MinMaxScaler()),  
 ('PCA',PCA(n\_components=3)),  
 ('classifier', LogisticRegression(solver ='sag')),  
])

12- Entrenar este modelo, predecir el conjunto de test y redondear las predicciones de forma inteligente. Explicar el criterio de redondeo.

In [46]:

%%time  
model.fit(X\_train, y\_train)

/home/isanti/anaconda3/lib/python3.7/site-packages/sklearn/linear\_model/logistic.py:460: FutureWarning: Default multi\_class will be changed to 'auto' in 0.22. Specify the multi\_class option to silence this warning.  
 "this warning.", FutureWarning)

CPU times: user 3min 36s, sys: 2.79 s, total: 3min 39s  
Wall time: 3min 35s

Out[46]:

Pipeline(memory=None,  
 steps=[('normalizer', MinMaxScaler(copy=True, feature\_range=(0, 1))), ('PCA', PCA(copy=True, iterated\_power='auto', n\_components=3, random\_state=None,  
 svd\_solver='auto', tol=0.0, whiten=False)), ('classifier', LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,  
 intercept\_scaling=1, max\_iter=100, multi\_class='warn',  
 n\_jobs=None, penalty='l2', random\_state=None, solver='sag',  
 tol=0.0001, verbose=0, warm\_start=False))])

In [47]:

y\_pred = model.predict(X\_test)

In [48]:

y\_train

Out[48]:

array([ 2, 5, 1, ..., 18, 10, 6])

No redondeo porque me dan valores enteros

13- Calcular las métricas ontime , delay y early, sin ventana.

In [49]:

metrics = {  
 'ontimesv': ontimesv(y\_test, y\_pred),  
 'delaysv': delaysv(y\_test, y\_pred),  
 'earlysv': earlysv(y\_test, y\_pred),  
}  
  
metrics

Out[49]:

{'ontimesv': 0.4793805340953187,  
 'delaysv': 0.37123857903834623,  
 'earlysv': 0.14938088686633505}

El 37 por ciento llego despues de la prediccion, el 15 por ciento antes y el 48 por ciento llego a tiempo. Un 63 por ciento llego antes o en el tiempo predicho. podemos decir que el modelo es bueno pero los dos anteriores dieron mejores metricas.

14- Justificar muy brevemente la elección del modelo.

Elegi Logistic Regression porque es facil de entender como funciona y nos da informacion de la probabilidad que tiene cada etiqueta en cada prediccion que se hizo lo que puede ser util para elegir el offset. Ademas me da informacion de que tan segura es posible estar de la prediccion hecha.

Ventanas de predicción: 15- Construir un offset para mejorar las predicciones de nuestros modelos, de forma que tenga avg\_offset menor o igual a 1, recalcular las métricas y explicar cómo se lo construyó.

In [50]:

**def** ontime(y\_test, lower\_bound, upper\_bound):  
 ontime\_msk = (lower\_bound <= y\_test) & (y\_test <= upper\_bound)  
 **return** np.sum(ontime\_msk) / np.size(y\_test)  
  
**def** delay(y\_test, lower\_bound, upper\_bound):  
 delay\_msk = (upper\_bound < y\_test)  
 **return** np.sum(delay\_msk) / np.size(y\_test)  
  
**def** early(y\_test, lower\_bound, upper\_bound):  
 early\_msk = (y\_test < lower\_bound)  
 **return** np.sum(early\_msk) / np.size(y\_test)  
  
**def** offset\_window(y\_test, lower\_bound, upper\_bound, length):  
 offset\_msk = ((upper\_bound - lower\_bound) == length)  
 **return** np.sum(offset\_msk) / np.size(offset\_msk)  
  
**def** avg\_speed(y\_test, lower\_bound, upper\_bound):  
 **return** lower\_bound.mean()  
  
**def** avg\_offset(y\_test, lower\_bound, upper\_bound):  
 **return** (upper\_bound - lower\_bound).mean()  
  
**def** get\_metrics(y\_test, speed, offset):  
 lower\_bound = speed  
 upper\_bound = speed + offset  
 metrics = {'on\_time': ontime(y\_test, lower\_bound, upper\_bound).astype(float).round(3),  
 'delay': delay(y\_test, lower\_bound, upper\_bound).astype(float).round(3),  
 'early': early(y\_test, lower\_bound, upper\_bound).astype(float).round(3),  
 'offset\_0': offset\_window(y\_test, lower\_bound, upper\_bound, 0).astype(float).round(3),  
 'offset\_1': offset\_window(y\_test, lower\_bound, upper\_bound, 1).astype(float).round(3),  
 'offset\_2': offset\_window(y\_test, lower\_bound, upper\_bound, 2).astype(float).round(3),  
 'avg\_speed': avg\_speed(y\_test, lower\_bound, upper\_bound).astype(float).round(3),  
 'avg\_offset': avg\_offset(y\_test, lower\_bound, upper\_bound).astype(float).round(3),  
 }  
  
 **return** metrics

In [51]:

pred\_prob = model.predict\_proba(X\_test)  
offset = np.zeros\_like(y\_pred)  
  
**for** i **in** range(len(y\_pred)):  
 **if** (pred\_prob[i].max() < 0.01):  
 offset[i]=3  
 **elif** (pred\_prob[i].max()>=0.01) & (pred\_prob[i].max()<0.1):  
 offset[i]=2  
 **elif** (pred\_prob[i].max()>=0.1) & (pred\_prob[i].max()<0.30):  
 offset[i]=1  
get\_metrics(y\_test, y\_pred, offset)

Out[51]:

{'on\_time': 0.671,  
 'delay': 0.18,  
 'early': 0.149,  
 'offset\_0': 0.221,  
 'offset\_1': 0.702,  
 'offset\_2': 0.077,  
 'avg\_speed': 1.531,  
 'avg\_offset': 0.857}

Viendo las probabilidades de cada etiqueta tomo el maximo valor y si es menor al 1 por ciento le asigno un offset de 3, si esta entre 1 y 10 por ciento le asigno un offset de 2 y si esta entre 10 por ciento y 30 por ciento le asigno un offset de 1 y finalmente, le asigno un cero de offset al resto.

16- Construir un offset que mejore las métricas de los modelo y que además tenga un avg\_offset menor o igual que 2.5, recalcular las métricas y explicar cómo se lo construyó.

In [52]:

pred\_prob = model.predict\_proba(X\_test)  
offset = np.zeros\_like(y\_pred)  
  
**for** i **in** range(len(y\_pred)):  
 **if** (pred\_prob[i].max() < 0.2):  
 offset[i]=3  
 **elif** (pred\_prob[i].max()>=0.2) & (pred\_prob[i].max()<0.3):  
 offset[i]=2  
 **elif** (pred\_prob[i].max()>=0.3) & (pred\_prob[i].max()<0.40):  
 offset[i]=1  
get\_metrics(y\_test, y\_pred, offset)

Out[52]:

{'on\_time': 0.811,  
 'delay': 0.04,  
 'early': 0.149,  
 'offset\_0': 0.003,  
 'offset\_1': 0.218,  
 'offset\_2': 0.484,  
 'avg\_speed': 1.531,  
 'avg\_offset': 2.072}

Viendo las probabilidades de cada etiqueta tomo el maximo valor y si es menor al 20 por ciento le asigno un offset de 3, si esta entre 20 y 30 por ciento le asigno un offset de 2 y si esta entre 30 por ciento y 40 por ciento le asigno un offset de 1 y finalmente, le asigno un cero de offset al resto.

In [ ]:

In [ ]: