

# PEC 3 DISEÑO E IMPLEMENTACIÓN DEL TRABAJO

**TFM GESTIÓN AVANZADA DE INVENTARIO.** Machine Learning aplicado a la predicción de la demanda.

## 3 Diseño modelos de predicción

Se han completado las primeras fases de preparación del fichero, análisis estadístico y temporal, y por último reducción de la dimensionalidad.

Se aborda en este apartado, el diseño de un modelo de predicción de unidades de venta, a partir de algoritmos Machine learning.

In [171...]

```
# Carga de librerías.

import numpy as np
import pandas as pd
import seaborn as sns
import datetime
from datetime import datetime, timedelta
from pandas import DataFrame, Timestamp

from sklearn import datasets
from sklearn import preprocessing
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix, classification_report, mean_absolute_error
from sklearn.metrics import accuracy_score, recall_score, f1_score, precision_score
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, StandardScaler, MinMaxScaler
from sklearn.svm import SVC, SVR
from sklearn.tree import DecisionTreeClassifier, export_graphviz, DecisionTreeRegressor
from sklearn.utils import check_random_state
from scipy import stats
from scipy.stats import norm

# Visualizar árboles
from IPython.display import Image
import pydotplus
from six import StringIO

# UMAP para la reducción de dimensionalidad
# import umap

# Visualización
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt

# visualizar heatmap
import seaborn as sn

# Validación cruzada
from sklearn.model_selection import StratifiedKFold

# Redes neuronales
import keras
from keras.models import Sequential, Model
from keras.layers import Dense, Activation, Flatten, Input
from keras.optimizers import Adam, SGD, Adadelta, Adagrad

%matplotlib inline
```

In [3]:

```
# Eliminación de los límites para visualización.

pd.options.display.max_rows = None
pd.options.display.max_columns = None
```

```
In [4]: # Lectura del fichero depurado y limpio.  
# Indexar por variable 'date'
```

```
venta = pd.read_excel('data/venta_4.xls')  
venta = venta.set_index('date')  
# venta.sort_index()  
# venta = venta.drop(columns = ['Unnamed: 0'], axis=1)
```

```
In [4]: # Se eliminan las filas creadas por hueco en fecha.
```

```
# venta = venta.drop(venta[venta['fecha']==1].index, axis = 0)
```

```
In [5]: venta.head()
```

```
Out[5]:
```

	sku	fecha	udsVenta	bolOpen	bolPromo	udsStock	day	week	year	monday
		date								
2020-05-19	1	20200519	35	1	0	441	19	21	0	0
2020-05-20	1	20200520	28	1	0	406	20	21	0	0
2020-05-21	1	20200521	63	1	0	378	21	21	0	0
2020-05-22	1	20200522	42	1	0	315	22	21	0	0
2020-05-23	1	20200523	28	1	0	273	23	21	0	0

```
In [6]: print ('Dimensión del fichero venta {}'.format(venta.shape))
```

Dimensión del fichero venta (32236, 47).

### 3.1 Discretizar variable objetivo udsVenta

Los algoritmos de Machine Learning, funcionan mejor en problemas de clasificación, donde la variable objetivo es discreta. En este ejercicio, la variable objetivo udsVenta es continua y debe transformarse en continua. Se discretiza según criterio de tamaño o frecuencia.

#### 3.1.1 Discretización en 7 intervalos de igual longitud.

Se crean rangos de manera que cada intervalo represente el mismo número de valores.

```
In [7]: print('El valor máximo de udsVenta es {} y el mínimo {}.'.format(venta['udsVenta']))
```

El valor máximo de udsVenta es 273 y el mínimo 0.

El valor máximo de la variable udsVenta es 273 y el mínimo 0. Se reparte el rango de valores, en 7 intervalos de igual tamaño.

```
In [8]: venta['udsVenta_cat'] = pd.cut(venta.udsVenta, bins=7, labels=np.arange(7), right=True)
```

```
In [9]: venta['udsVenta_cat'].describe()
```

```
Out[9]: count    32236
unique      7
top        0
freq     29718
Name: udsVenta_cat, dtype: int64
```

```
In [10]: freq_cat = venta['udsVenta_cat'].value_counts()
freq_cat
```

```
Out[10]: 0    29718
1    2320
2    163
3    28
4    3
6    2
5    2
Name: udsVenta_cat, dtype: int64
```

La distribución de clases está muy sesgada hacia la derecha. El 81% de las observaciones corresponde a la clase 0.

Se prueba con 14 intervalos.

### 3.1.2 Discretización en 14 intervalos de igual longitud.

Se amplía el número de intervalos a 14, que también es múltiplo de 196.

```
In [11]: venta['udsVenta_cat2'] = pd.cut(venta.udVenta, bins=14, labels=np.arange(14), ri
```

```
In [12]: venta['udsVenta_cat2'].describe()
```

```
Out[12]: count    32236
unique      14
top        0
freq     22056
Name: udsVenta_cat2, dtype: int64
```

```
In [13]: freq_cat = venta['udsVenta_cat2'].value_counts()
freq_cat
```

```
Out[13]: 0    22056
1    7662
2    1860
3    460
4    108
5    55
6    20
7     8
8     2
13    1
12    1
11    1
10    1
9     1
Name: udsVenta_cat2, dtype: int64
```

El intervalo 0, acumula el 55% de las observaciones

### 3.1.3 Discretización por valores de *udsVenta* (clases).

Vease un segundo método de discretización consistente en crear intervalos para cada uno de

```
In [14]: freq = venta['udsVenta'].value_counts()  
freq
```

```
Out[14]: 0      11096  
7       5902  
14      5058  
21      3767  
28      2396  
35      1499  
42       948  
49       570  
56       342  
63       224  
70       143  
77        93  
84        63  
91        45  
98        27  
105      15  
112      13  
119        9  
126        8  
147        5  
140        3  
133        3  
161        2  
196        1  
252        1  
273        1  
182        1  
224        1  
Name: udsVenta, dtype: int64
```

El valor de venta = 0 es el 32,74% de las observaciones. Sólo interesa incorporar al estudio aquellas observaciones en las que el valor de venta = 0, estando abierto el establecimiento. Por tanto, vease cómo resultaría el reparto de frecuencias para unidades de venta, cuando el establecimiento está abierto *bolOpen* = 1.

```
In [16]: venta_open = venta[venta['bolOpen']==1]  
freq_open = venta_open['udsVenta'].value_counts()  
freq_open
```

```
Out[16]: 0      6934  
7       5896  
14      5053  
21      3766  
28      2396  
35      1499  
42       947  
49       569  
56       341  
63       224  
70       143  
77        93  
84        63  
91        45  
98        27  
105      15  
112      13  
119        9
```

```
126      8
147      5
140      3
133      3
161      2
196      1
252      1
273      1
182      1
224      1
Name: udsVenta, dtype: int64
```

Se han descartado un total de 4.177 observaciones en las que el establecimiento está cerrado. Ahora, el valor de ventas = 0, representa el 23,33% del total.

```
In [17]: # Ordenado por valores de venta
freq_open.sort_index()
```

```
Out[17]: 0      6934
7      5896
14     5053
21     3766
28     2396
35     1499
42     947
49     569
56     341
63     224
70     143
77     93
84     63
91     45
98     27
105    15
112    13
119    9
126    8
133    3
140    3
147    5
161    2
182    1
196    1
224    1
252    1
273    1
Name: udsVenta, dtype: int64
```

Hay un total de 25 valores distintos en *udsVenta*. Se crea variable clase para cada valor de *udsVenta*. Todos son multiplos de 7, excepto 1. Se asigna clase a la variable *\_udsVentaclass*.

In [18]:

```
clas = venta['udsVenta'].unique().tolist()
clas.sort()
K = range (0, len(clas))

venta = venta.reset_index()
venta['udsVenta_clas'] = 0
j=0

table = pd.DataFrame(columns =('udsVenta', 'clas'))

for k in K:
    c = clas[k]
    V = venta.index[venta['udsVenta'] == c].tolist()
    venta.at[V, 'udsVenta_clas'] = j
    table.loc[k,'udsVenta'] = c
    table.loc[k, 'clas'] = j
    j=j+1

venta = venta.set_index('date')
```

In [19]:

```
# Tabla con asignación de clases a la variable udsVenta.

table.astype('int32')
```

Out[19]:

	udsVenta	clas
<b>0</b>	0	0
<b>1</b>	7	1
<b>2</b>	14	2
<b>3</b>	21	3
<b>4</b>	28	4
<b>5</b>	35	5
<b>6</b>	42	6
<b>7</b>	49	7
<b>8</b>	56	8
<b>9</b>	63	9
<b>10</b>	70	10
<b>11</b>	77	11
<b>12</b>	84	12
<b>13</b>	91	13
<b>14</b>	98	14
<b>15</b>	105	15
<b>16</b>	112	16
<b>17</b>	119	17
<b>18</b>	126	18
<b>19</b>	133	19

	udsVenta	clas
20	140	20
21	147	21
22	161	22
23	182	23
24	196	24
25	224	25
26	252	26

Este método de discretización no es representativo de todos los posibles valores de la variable *udsVenta*, dado que no contempla los valores (154, 161, 168, 175, 189). En cambio, tiene la ventaja que separa el valor *udsVenta* = 1, del primer rango, y reduce la concentración del valor 0.

In [19]:

```
# Cambiar categoria a la nueva variable.

venta['udsVenta_clas'] = venta['udsVenta'].astype('category')
venta.dtypes
```

Out[19]:

sku	int64
fecha	int64
udsVenta	int64
bolOpen	int64
bolPromo	int64
udsStock	int64
day	int64
week_year	int64
monday	int64
tuesday	int64
wednesday	int64
thursday	int64
friday	int64
saturday	int64
sunday	int64
month_1	int64
month_2	int64
month_3	int64
month_4	int64
month_5	int64
month_6	int64
month_7	int64
8	int64
9	int64
10	int64
11	int64
12	int64
2020	int64
2021	int64
2022	int64
bol_weekfirst	int64
bol_weeklast	int64
udsVentalog	float64
bol_festivo	int64
bol_puente	int64
bol_week_prev1	int64
bol_week_prev2	int64
bol_week_post1	int64

```

bol_week_post2      int64
bol_day_prev1       int64
bol_day_prev2       int64
bol_day_post1       int64
bol_day_post2       int64
bol_summer1q        int64
bol_summer2q        int64
bol_summer3q        int64
bol_summer4q        int64
udsVenta_cat        category
udsVenta_cat2       category
udsVenta_clas       category
dtvne: object

```

In [20]:

```
# Nuevo fichero para días que está abierto y categoría por rango de unidades vendidas
venta.head()
```

Out[20]:

	sku	fecha	udsVenta	bolOpen	bolPromo	udsStock	day	week_year	monday
	date								
2020-05-19	1	20200519	35	1	0	441	19	21	0
2020-05-20	1	20200520	28	1	0	406	20	21	0
2020-05-21	1	20200521	63	1	0	378	21	21	0
2020-05-22	1	20200522	42	1	0	315	22	21	0
2020-05-23	1	20200523	28	1	0	273	23	21	0

### 3.2 Reducción fichero para días con establecimiento abierto (*bolOpen* = 1).

Se encuentran razones por las que pensar que los días que establecimiento está cerrado, no aporta información relevante al fichero. En estos días el volumen de ventas es 0.

In [21]:

```
venta2 = venta[venta['bolOpen']==1]
```

In [22]:

```
venta2.head()
```

Out[22]:

	sku	fecha	udsVenta	bolOpen	bolPromo	udsStock	day	week_year	monday
	date								
2020-05-19	1	20200519	35	1	0	441	19	21	0
2020-05-20	1	20200520	28	1	0	406	20	21	0
2020-05-21	1	20200521	63	1	0	378	21	21	0
2020-05-22	1	20200522	42	1	0	315	22	21	0
2020-05-23	1	20200523	28	1	0	273	23	21	0

In [23]:

```
freq_sku = venta_open.groupby(['sku', 'udsVenta']).size()
```

### 3.3 Conjuntos train y test

Se separan los ficheros, 80% conjunto train y 20% conjunto test.

In [24]:

```
# Conjunto train con 542 observaciones que equivale al 80%.  
print('Fecha máxima para fichero train {}'.format(venta.iloc[519,1]))
```

Fecha máxima para fichero train 20211114.

In [25]:

```
# Duplicar la variable sku, y añadirla como índice del fichero venta  
venta['id'] = venta['sku']  
venta = venta.set_index('id', append=True)
```

In [26]:

```
# Duplicar la variable sku, y añadirla como índice del fichero venta2  
venta2['id'] = venta2['sku']  
venta2 = venta2.set_index('id', append=True)
```

In [27]:

```
# Conjunto train y test para todo el fichero  
  
train = venta.loc[venta['fecha'] <= 20211021]  
test = venta.loc[venta['fecha'] > 20211021]  
  
train2 = venta2.loc[venta2['fecha'] <= 20211021]  
test2 = venta2.loc[venta2['fecha'] > 20211021]
```

In [28]:

```
train.head()
```

Out[28]:

	sku	fecha	udsVenta	bolOpen	bolPromo	udsStock	day	week_year	monday
	date	id							
2020-05-19	1	1	20200519	35	1	0	441	19	21
2020-05-20	1	1	20200520	28	1	0	406	20	21
2020-05-21	1	1	20200521	63	1	0	378	21	21
2020-05-22	1	1	20200522	42	1	0	315	22	21
2020-05-23	1	1	20200523	28	1	0	273	23	21

In [29]:

```
train2.head()
```

Out[29]:

	sku	fecha	udsVenta	bolOpen	bolPromo	udsStock	day	week_year	monday
	date	id							
2020-05-19	1	1	20200519	35	1	0	441	19	21
2020-05-20	1	1	20200520	28	1	0	406	20	21
2020-05-21	1	1	20200521	63	1	0	378	21	21
2020-05-22	1	1	20200522	42	1	0	315	22	21
2020-05-23	1	1	20200523	28	1	0	273	23	21

In [32]:

```
print('Dimensión del fichero train {}'.format(train.shape))
print('Dimensión del fichero test {}'.format(test.shape))
print('El fichero train es un {}% del fichero total.'.format(round(len(train)/len(t
```

Dimensión del fichero train (25607, 50).

Dimensión del fichero test (6629, 50).

El fichero train es un 79.44% del fichero total.

In [31]:

```
print('Dimensión del fichero train {}'.format(train2.shape))
print('Dimensión del fichero test {}'.format(test2.shape))
print('El fichero train es un {}% del fichero total.'.format(round(len(train2)/len(t
```

Dimensión del fichero train (22033, 50).

Dimensión del fichero test (6026, 50).

El fichero train es un 78.52% del fichero total.

In [33]:

```
# Separación de la variable dependiente e independientes para el fichero completo.

y_train_1 = train['udsVenta']
y_train_2 = train['udsVenta_cat']
y_train_3 = train['udsVenta_clas']
y_train_4 = train['udsVenta_cat2']
x_train = train.drop(columns = ['udsVenta', 'udsVentalog', 'udsVenta_cat', 'udsVenta_clas', 'fecha', 'udsVenta_clas'], axis = 1)

y_test_1 = test['udsVenta']
y_test_2 = test['udsVenta_cat']
y_test_3 = test['udsVenta_clas']
y_test_4 = test['udsVenta_cat2']
x_test = test.drop(columns = ['udsVenta', 'udsVentalog', 'udsVenta_cat', 'udsVenta_clas', 'fecha', 'udsVenta_clas'], axis = 1)
```

In [34]:

y\_test\_1.shape

Out[34]:

(6629,)

In [35]:

```
# Separación de la variable dependiente e independientes para el fichero días abiertos.

y_train2_1 = train2['udsVenta']
y_train2_2 = train2['udsVenta_cat']
y_train2_3 = train2['udsVenta_clas']
y_train2_4 = train2['udsVenta_cat2']
x_train2 = train2.drop(columns = ['udsVenta', 'udsVentalog', 'udsVenta_cat', 'udsVenta_clas', 'fecha', 'udsVenta_clas'], axis = 1)

y_test2_1 = test2['udsVenta']
y_test2_2 = test2['udsVenta_cat']
y_test2_3 = test2['udsVenta_clas']
y_test2_4 = test2['udsVenta_cat2']
x_test2 = test2.drop(columns = ['udsVenta', 'udsVentalog', 'udsVenta_cat', 'udsVenta_clas', 'fecha', 'udsVenta_clas'], axis = 1)
```

In [36]:

y\_test2\_1.shape

Out[36]:

(6026,)

In [37]:

```
# Normalización de las variables independientes en rango [0, 1]

# x_train = (x_train-x_train.min())/(x_train.max()-x_train.min())
# x_test = (x_test-x_test.min())/(x_test.max()-x_test.min())
# x_train = x_train.fillna(0)
# x_test = x_test.fillna(0)
```

In [38]:

```
# X_train = np.asarray(x_train)
# X_test = np.asarray(x_test)
# Y_train = np.asarray(y_train)
# Y_test = np.asarray(y_test)
```

In [37]:

```
# Modifica variable continua a discreta
```

```
label_encoder = LabelEncoder()
y_train_class = label_encoder.fit_transform(y_train_1)
y_test_class = label_encoder.fit_transform(y_test_1)
```

### 3.3 Agrupamiento por la variable sku.

Se lanza el algoritmo para cada uno de los valores de sku. Se intuye que el modelo obtenido dará mayor precisión cuando se entrena por separado por cada sku, puesto que el modelo de venta para cada uno de ellos, no necesariamente debe ser el mismo.

In [40]:

```
# Separación del data set venta en sub ficheros según el valor de la variable sku.

# sku = venta['sku'].unique().tolist()
# K = range (0, len(sku))
# data_out = pd.DataFrame()
# i = 0

def sep_sku(data, sku):
    # Se separa del fichero data, las observaciones para el valor de sku.
    df = data[data['sku']==sku]

    # Separación de los conjuntos train y test
    dft = df.loc[df['fecha']<= 20211021]
    dfs = df.loc[df['fecha']> 20211021]
    y_train_1_sku = dft['udsVenta']
    y_train_2_sku = dft['udsVenta_cat']
    y_train_3_sku = dft['udsVenta_clas']
    y_train_4_sku = dft['udsVenta_cat2']
    x_train_sku = dft.drop(columns = ['udsVenta', 'udsVentalog', 'udsVenta_cat',
                                      'fecha', 'udsVenta_clas'], axis = 1)
    y_test_1_sku = dfs['udsVenta']
    y_test_2_sku = dfs['udsVenta_cat']
    y_test_3_sku = dfs['udsVenta_clas']
    y_test_4_sku = dfs['udsVenta_cat2']
    x_test_sku = dfs.drop(columns = ['udsVenta', 'udsVentalog', 'udsVenta_cat',
                                      'fecha', 'udsVenta_clas'], axis = 1)
    return (y_train_1_sku, y_train_2_sku, y_train_3_sku, y_train_4_sku, x_train_sku,
            y_test_2_sku, y_test_3_sku, y_test_4_sku, x_test_sku)
```

### 3.4 Algoritmos Machine Learning.

Vease en este apartado cada uno de los algoritmos, aplicados al data set venta, y a cada uno de los subficheros obtenidos por agrupamiento de la variable sku.

### 3.4.1 Modelo Gausiano

#### 3.4.1.1 Modelo Gausiano para todos los días, sin filtrar por variable bolOpen.

Se entrena en este apartado un modelo Gausiano en el que no se consideran todos los días, incluso los que el establecimiento está cerrado. La variable objetivo *udsVenta* está categorizada en 7 intervalos de igual tamaño.

In [200...]

```
model_bayes2 = GaussianNB()
model_bayes2.fit(x_train, y_train_2)
```

Out[200...]

```
GaussianNB()
```

In [201...]

```
# Precisión sobre train y test
score_train = model_bayes2.score(x_train, y_train_2)
print('El grado de precisión del modelo para el conjunto train es {}'.format(round(score_train, 4)))
score_test = model_bayes2.score(x_test, y_test_2)
print('El grado de precisión del modelo para el conjunto test es {}'.format(round(score_test, 4)))
```

El grado de precisión del modelo para el conjunto train es 0.271.  
El grado de precisión del modelo para el conjunto test es 0.4666.

In [202...]

```
# Predicción del modelo
train_preds_NB2 = model_bayes2.predict(x_train)
test_preds_NB2 = model_bayes2.predict(x_test)

#Vemos el error del algoritmo a la hora de predecir, para comprobar si a ajustado
print('MAE in train:', mean_absolute_error(train_preds_NB2, y_train_2))
print('RMSE in train:', np.sqrt(mean_squared_error(train_preds_NB2, y_train_2)))
print('MAE in test:', mean_absolute_error(test_preds_NB2, y_test_2))
print('RMSE in test:', np.sqrt(mean_squared_error(test_preds_NB2, y_test_2)))
```

MAE in train: 1.7587768969422424  
RMSE in train: 2.190278759662055  
MAE in test: 1.0191582440790465  
RMSE in test: 1.5551412551364472

In [203...]

```
# Precisión, recall y f1 del modelo en conjunto train y test

accuracy_train = accuracy_score(y_train_2, train_preds_NB2, normalize = True)
accuracy_test = accuracy_score(y_test_2, test_preds_NB2, normalize = True)
print('Accuracy train: {}, Accuracy test: {}'.format(round(accuracy_train, 4), round(accuracy_test, 4)))

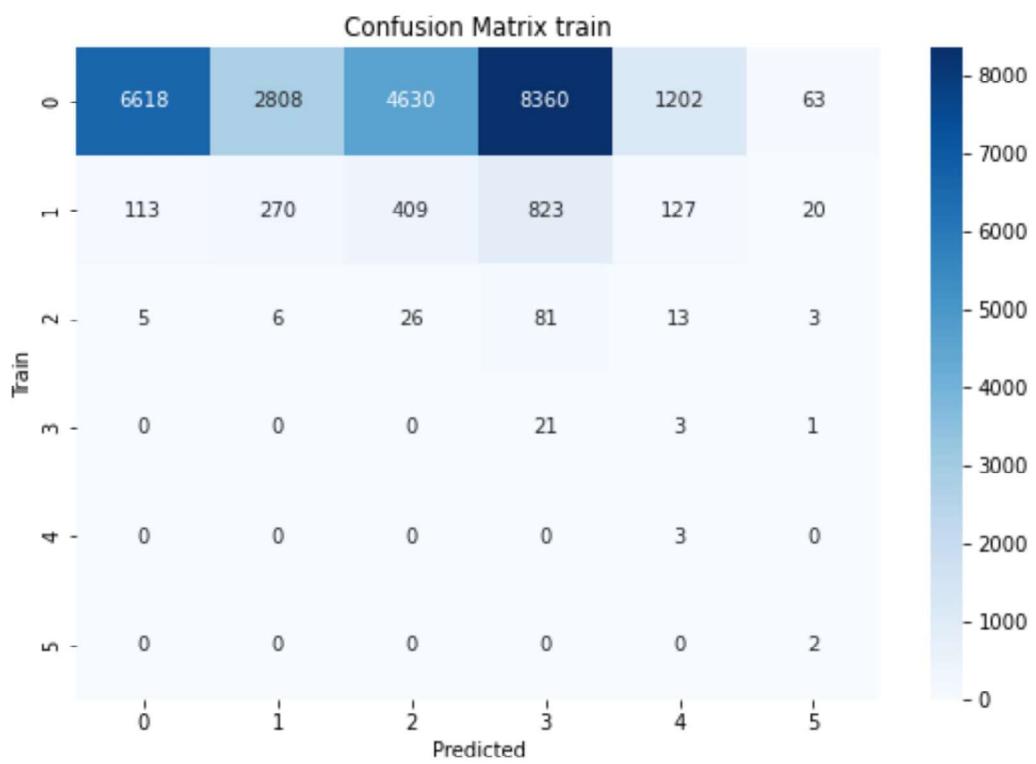
recall_train = recall_score(y_train_2, train_preds_NB2, average= 'weighted')
recall_test = recall_score(y_test_2, test_preds_NB2, average= 'weighted')
print('Recall train: {}, Recall test: {}'.format(round(recall_train, 4), round(recall_test, 4)))

f1_train = f1_score(y_train_2, train_preds_NB2, average= 'weighted')
f1_test = f1_score(y_test_2, test_preds_NB2, average= 'weighted')
print('F1 train: {}, F1 test: {}'.format(round(f1_train, 4), round(f1_test, 4)))
```

Accuracy train: 0.271, Accuracy test: 0.4666  
Recall train: 0.271, Recall test: 0.4666  
F1 train: 0.4102, F1 test: 0.5934

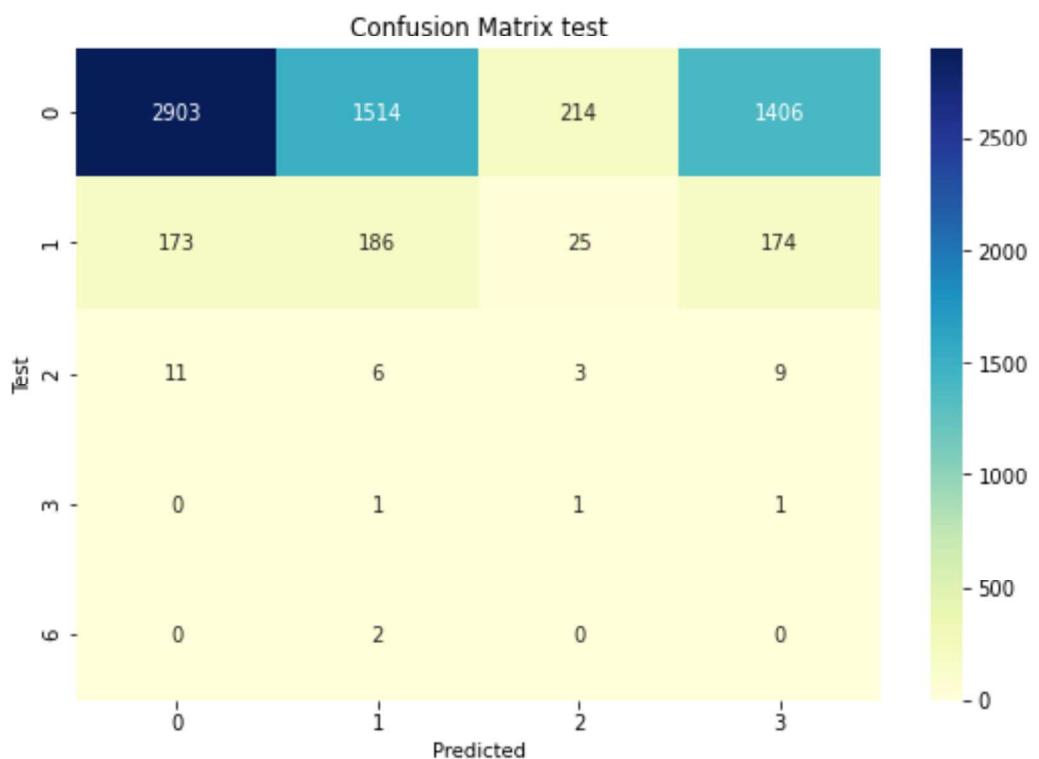
In [204...]

```
# Representación gráfica de la matriz de confusión sobre el conjunto train.  
confusion_matrix_table = pd.crosstab(y_train_2, train_preds_NB2, rownames=['Train'])  
plt.figure(1, figsize=(9, 6))  
sn.heatmap(confusion_matrix_table, annot=True, cmap="Blues", fmt='g')  
plt.title("Confusion Matrix train")  
plt.savefig('graph/matriz_confusion_NB2train_2.jpg')  
plt.show()
```



In [205...]

```
# Representación gráfica de la matriz de confusión sobre el conjunto test.  
confusion_matrix_table = pd.crosstab(y_test_2, test_preds_NB2, rownames=['Test'])  
plt.figure(1, figsize=(9, 6))  
sn.heatmap(confusion_matrix_table, annot=True, cmap="YlGnBu", fmt='g')  
plt.title("Confusion Matrix test")  
plt.savefig('graph/matriz_confusion_NB2test_2.jpg')  
plt.show()
```



In [206...]

```
y_predict = pd.read_excel('data/y_predictsvm.xls')
test_preds_NB2 = test_preds_NB2.astype(int)
y_predict['test_preds_NB2'] = test_preds_NB2
y_predict.to_excel('data/y_predictsvm.xls')
```

### 3.4.1.2 Modelo Gausiano, sólo para días de apertura.

Se entrena el modelo para el conjunto de sólo los días de apertura, y variable *udsVenta* categorizada.

In [41]:

```
# Implementación clasificador Gaussian Naïve Bayes
model_bayes = GaussianNB()
model_bayes.fit(x_train2, y_train2_2)
```

Out[41]: GaussianNB()

In [42]:

```
# Precisión sobre train y test
score_train = model_bayes.score(x_train2, y_train2_2)
print('El grado de precisión del modelo para el conjunto train es {}.'.format(round(score_train, 4)))
score_test = model_bayes.score(x_test2, y_test2_2)
print('El grado de precisión del modelo para el conjunto test es {}.'.format(round(score_test, 4)))
```

El grado de precisión del modelo para el conjunto train es 0.3507.  
El grado de precisión del modelo para el conjunto test es 0.5541.

In [43]:

```
# Predicción del modelo
train_preds_NB = model_bayes.predict(x_train2)
test_preds_NB = model_bayes.predict(x_test2)

#Vemos el error del algoritmo a la hora de predecir, para comprobar si a ajustado
print('MAE in train:', mean_absolute_error(train_preds_NB, y_train2_2))
print('RMSE in train:', np.sqrt(mean_squared_error(train_preds_NB, y_train2_2)))
print('MAE in test:', mean_absolute_error(test_preds_NB, y_test2_2))
print('RMSE in test:', np.sqrt(mean_squared_error(test_preds_NB, y_test2_2)))
```

```
MAE in train: 1.7079834793264648
RMSE in train: 2.2257024040501747
MAE in test: 0.9216727514105543
RMSE in test: 1.5220472875601467
```

In [44]:

```
# Precisión, recall y f1 del modelo en conjunto train y test

accuracy_train = accuracy_score(y_train2_2, train_preds_NB, normalize = True)
accuracy_test = accuracy_score(y_test2_2, test_preds_NB, normalize = True)
print('Accuracy train: {}, Accuracy test: {}'.format(round(accuracy_train, 4), round(accuracy_test, 4)))

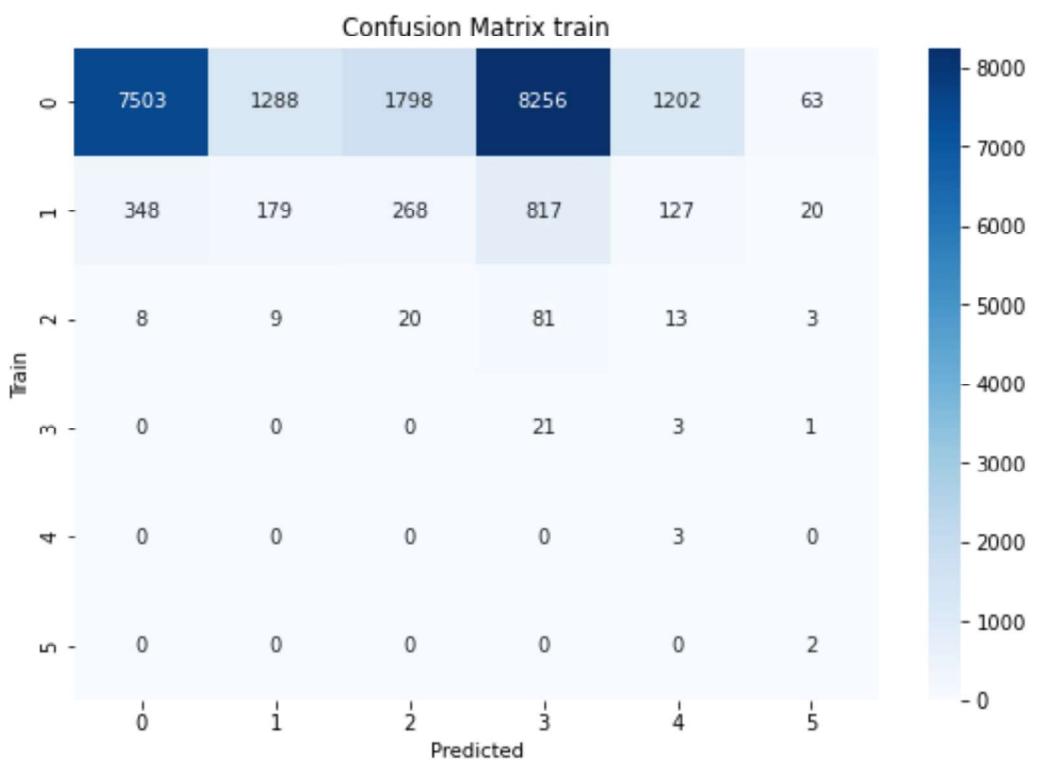
recall_train = recall_score(y_train2_2, train_preds_NB, average= 'weighted')
recall_test = recall_score(y_test2_2, test_preds_NB, average= 'weighted')
print('Recall train: {}, Recall test: {}'.format(round(recall_train, 4), round(recall_test, 4)))

f1_train = f1_score(y_train2_2, train_preds_NB, average= 'weighted')
f1_test = f1_score(y_test2_2, test_preds_NB, average= 'weighted')
print('F1 train: {}, F1 test: {}'.format(round(f1_train, 4), round(f1_test, 4)))
```

```
Accuracy train: 0.3507, Accuracy test: 0.5541
Recall train: 0.3507, Recall test: 0.5541
F1 train: 0.4987, F1 test: 0.6654
```

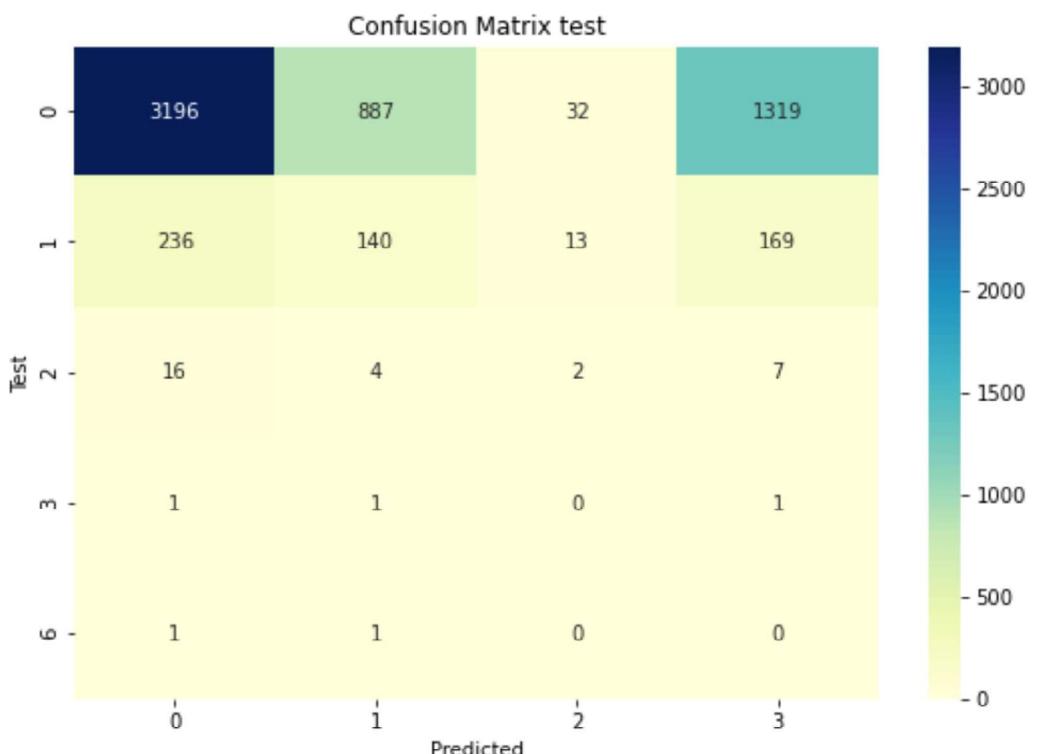
In [45]:

```
# Representación gráfica de la matriz de confusión sobre el conjunto train.
confusion_matrix_table = pd.crosstab(y_train2_2, train_preds, rownames=['Train'],
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="Blues", fmt='g')
plt.title("Confusion Matrix train")
plt.savefig('graph/matriz_confusion_NBtrain2_2.jpg')
plt.show()
```



In [46]:

```
# Representación gráfica de la matriz de confusión sobre el conjunto test.
confusion_matrix_table = pd.crosstab(y_test2_2, test_preds, rownames=['Test'], colnames=['Predicted'])
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="YlGnBu", fmt='g')
plt.title("Confusion Matrix test")
plt.savefig('graph/matriz_confusion_NBtest2_2.jpg')
plt.show()
```



### 3.4.2 Modelo SVM

Se aborda en este apartado el algoritmo SVM, para las siguientes configuraciones;

- Variable *udsVenta* como variable cuantitativa.
- Discretiza en 7 intervalos de igual tamaño (fichero sufijo 2).
- Discretiza en 25 clases que corresponde a los valores de sku (fichero sufijo 3).
- Discretiza en 14 intervalos de igual tamaño (fichero sufijo 4).

### 3.4.2.1 Configuración de parámetros de modelo SVM para valor determinado de sku.

Se busca la mejor configuración de parámetros para el modelo SVM, especialmente el kernel óptimo (lineal, logarítmico o polinomial). Aplicar GridSearchCV a todo el fichero pero demora mucho tiempo. Se decide configurar para una partición del fichero, para valor aleatorio de

In [47]:

```
df = venta[venta['sku']==23]

# Separación de Los conjuntos train y test
dft = df.loc[df['fecha']<= 20211021]
dfs = df.loc[df['fecha']> 20211021]

y_train_4_sku = dft['udsVenta_cat2']
x_train_sku = dft.drop(columns = ['udsVenta', 'udsVentalog', 'udsVenta_cat', 'udsVenta_clas', 'fecha', 'udsVenta_clas'], axis = 1)

y_test_4_sku = dfs['udsVenta_cat2']
x_test_sku = dfs.drop(columns = ['udsVenta', 'udsVentalog', 'udsVenta_cat', 'udsVenta_clas', 'fecha', 'udsVenta_clas'], axis = 1)
```

In [48]:

```
# Definición de Los conjuntos parámetro
kernel = ['linear', 'poly', 'rbf']
C = [2, 5, 10, 15]
cache_size = [100, 300, 500, 800]

grid_tree = dict(kernel=kernel, C=C, cache_size=cache_size)
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=15)

# Optimizar
SVM_optimizer = GridSearchCV(estimator=SVC(),
                             param_grid=grid_tree,
                             n_jobs=-1,
                             scoring='accuracy',
                             refit = True,
                             cv=cv,
                             error_score=0)
```

In [49]:

```
grid_SVM_optimizer = SVM_optimizer.fit(x_train_sku, y_train_4_sku)
```

```
C:\Users\jh100\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:668:
UserWarning: The least populated class in y has only 1 members, which is less than
n_splits=5.
% (min_groups, self.n_splits)), UserWarning)
```

In [50]:

```
# Obtener La mejor combinación del clasificador SVM
print ('La mejor configuración de parámetros que maximiza la precisión es {}.'.format(grid_SVM_optimizer.C))
print ('Clases de clasificación del modelo {}.'.format(grid_SVM_optimizer.classes_))

# Obtenemos La precisión del modelo
print('Precisión del modelo para el conjunto train {}'.format(round(grid_SVM_optimizer.score(x_train_sku, y_train_4_sku),4)))
print('Precisión del modelo para el conjunto test {}'.format(round(grid_SVM_optimizer.score(x_test_sku, y_test_4_sku),4)))
```

La mejor configuración de parámetros que maximiza la precisión es {'C': 2, 'cache\_size': 100, 'kernel': 'poly'}.  
 Clases de clasificación del modelo [0 1 2 3].  
 Precisión del modelo para el conjunto train 0.7893.  
 Precisión del modelo para el conjunto test 0.6269.

Se entrena ahora el modelo para el fichero con fechas de apertura bolOpen=1 y se comparan las precisiones en ambos modelos.

In [51]:

```
df = venta2[venta2['sku']==23]

# Separación de los conjuntos train y test
dft = df.loc[df['fecha']<= 20211021]
dfs = df.loc[df['fecha']> 20211021]

y_train2_4_sku = dft['udsVenta_cat2']
x_train2_sku = dft.drop(columns = ['udsVenta', 'udsVentalog', 'udsVenta_cat', 'udsVenta_cat2', 'fecha', 'udsVenta_clas'], axis = 1)

y_test2_4_sku = dfs['udsVenta_cat2']
x_test2_sku = dfs.drop(columns = ['udsVenta', 'udsVentalog', 'udsVenta_cat', 'udsVenta_cat2', 'fecha', 'udsVenta_clas'], axis = 1)
```

In [52]:

```
grid2_SVM_optimizer = SVM_optimizer.fit(x_train2_sku, y_train2_4_sku)
```

```
C:\Users\jh100\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:668:
UserWarning: The least populated class in y has only 1 members, which is less than n_splits=5.
% (min_groups, self.n_splits)), UserWarning)
```

In [53]:

```
# Obtener La mejor combinación del clasificador SVM
print ('La mejor configuración de parámetros que maximiza la precisión es {}.'.format(grid2_SVM_optimizer.C))
print ('Clases de clasificación del modelo {}.'.format(grid2_SVM_optimizer.classes_))

# Obtenemos La precisión del modelo
print('Precisión del modelo para el conjunto train {}'.format(round(grid2_SVM_optimizer.score(x_train2_sku, y_train2_4_sku),4)))
print('Precisión del modelo para el conjunto test {}'.format(round(grid2_SVM_optimizer.score(x_test2_sku, y_test2_4_sku),4)))
```

La mejor configuración de parámetros que maximiza la precisión es {'C': 2, 'cache\_size': 100, 'kernel': 'poly'}.  
 Clases de clasificación del modelo [0 1 2 3].  
 Precisión del modelo para el conjunto train 0.7613.  
 Precisión del modelo para el conjunto test 0.6212.

Se ha comparado la precisión para el fichero de todas las fechas, con el de sólo días establecimiento está abierto. Se obtiene mayor precision con el fichero de todas las fechas. Entre todos los parámetros posibles, el que maximiza la precisión es C=2, kernel='poly' y

cache\_size=100.

Se entrena el algoritmo para las cuatro configuraciones antes descritas.

### 3.4.2.2 Modelo SVR para variable cuantitativa.

Se entrena modelo SVM con la variable objetivo *udsVenta* como variable cuantitativa.

In [54]:

```
# Modelo clasificación para variable dependiente continua
# SVR (versión del modelo SVM)
# Hiperplano lineal

svr = SVR(kernel='poly', C=2, cache_size=100)

# Entrenamiento
svr.fit(x_train, y_train_class)

# Predicción del modelo
train_preds_1 = svr.predict(x_train)
test_preds_1 = svr.predict(x_test)

#Vemos el error del algoritmo a la hora de predecir, para comprobar si a ajustado
print('MAE in train:', mean_absolute_error(train_preds_1, y_train_class))
print('RMSE in train:', np.sqrt(mean_squared_error(train_preds_1, y_train_class)))
print('MAE in test:', mean_absolute_error(test_preds_1, y_test_class))
print('RMSE in test:', np.sqrt(mean_squared_error(test_preds_1, y_test_class)))
```

MAE in train: 1.5954841903324533

RMSE in train: 2.3682287133355433

MAE in test: 1.7365111550938452

RMSE in test: 2.4957980874179486

In [55]:

```
# Precisión sobre train y test
score_train = svr.score(x_train, y_train_class)
print('El grado de precisión del modelo para el conjunto train es {}'.format(round(score_train, 4)))
score_test = svr.score(x_test, y_test_class)
print('El grado de precisión del modelo para el conjunto test es {}'.format(round(score_test, 4)))
```

El grado de precisión del modelo para el conjunto train es -0.0199.

El grado de precisión del modelo para el conjunto test es -0.127.

In [57]:

```
# Categorizar variable predicción para cálculo de Los parámetros del modelo.

label_encoder = LabelEncoder()

train_preds_class = label_encoder.fit_transform(train_preds_1)
test_preds_class = label_encoder.fit_transform(test_preds_1)
```

In [58]:

```
# Precisión, recall y f1 del modelo en conjunto train y test

accuracy_train = accuracy_score(y_train_class, train_preds_class, normalize = True)
accuracy_test = accuracy_score(y_test_class, test_preds_class, normalize = True)
print('Accuracy train: {}, Accuracy test: {}'.format(round(accuracy_train, 4), round(accuracy_test, 4)))

recall_train = recall_score(y_train_class, train_preds_class, average= 'weighted')
recall_test = recall_score(y_test_class, test_preds_class, average= 'weighted')
print('Recall train: {}, Recall test: {}'.format(round(recall_train, 4), round(recall_test, 4)))

f1_train = f1_score(y_train_class, train_preds_class, average= 'weighted')
f1_test = f1_score(y_test_class, test_preds_class, average= 'weighted')
print('F1 train: {}, F1 test: {}'.format(round(f1_train, 4), round(f1_test, 4)))
```

Accuracy train: 0.0, Accuracy test: 0.0002

Recall train: 0.0, Recall test: 0.0002

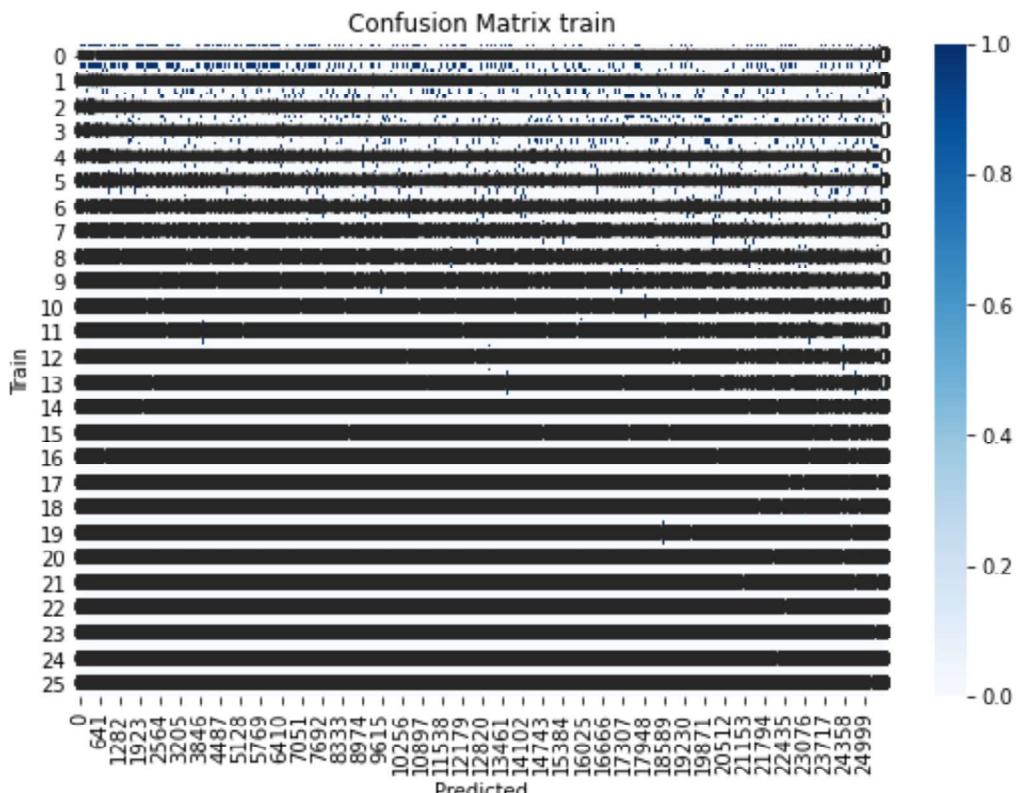
F1 train: 0.0001, F1 test: 0.0003

```
C:\Users\jh100\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:124
5: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels wi
th no true samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
C:\Users\jh100\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:124
5: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels wi
th no true samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
```

In [59]:

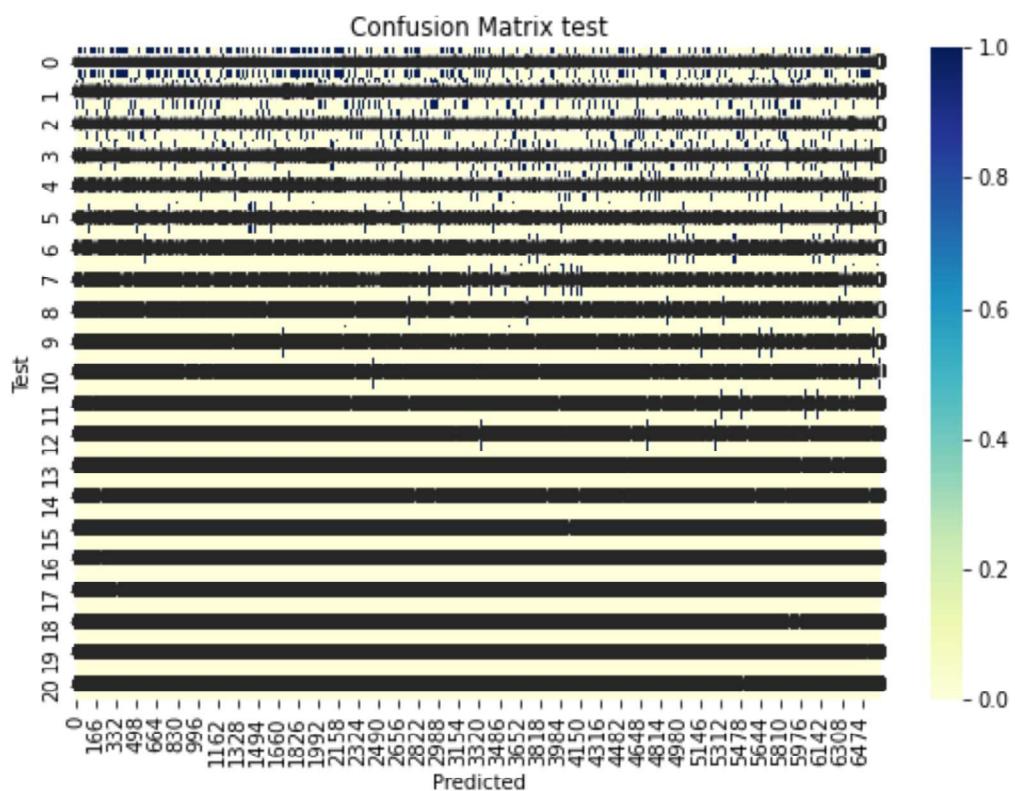
```
# Representación gráfica de la matriz de confusión sobre el conjunto train.

confusion_matrix_table = pd.crosstab(y_train_class, train_preds_class, rownames=[],
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="Blues", fmt='g')
plt.title("Confusion Matrix train")
plt.savefig('graph/matriz_confusion_SVCpolytrain_1.jpg')
plt.show()
```



In [60]:

```
# Representación gráfica de la matriz de confusión sobre el conjunto test.
confusion_matrix_table = pd.crosstab(y_test_class, test_preds_class, rownames=[ 'Test', 'Predicted'])
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="YlGnBu", fmt='g')
plt.title("Confusion Matrix test")
plt.savefig('graph/matriz_confusion_SVCpolytest_1.jpg')
plt.show()
```



### 3.4.2.3 Modelo SVM para variable *udsVenta* categorizada por intervalos de igual tamaño.

#### 3.4.2.3.1 Variable *udsVenta* discretizada en 7 intervalos de igual tamaño.

In [61]:

```
# Modelo clasificación para variable dependiente discreta
# SVC (versión del modelo SVM)
# Variable udsVenta categorizada en 7 clases

svm = SVC(kernel='poly', C=2, cache_size=100)

# Entrenamiento
svm.fit(x_train, y_train_2)

# Predicción del modelo
train_preds_2 = svm.predict(x_train)
test_preds_2 = svm.predict(x_test)

#Vemos el error del algoritmo a la hora de predecir, para comprobar si a ajustado
print('MAE in train:', mean_absolute_error(train_preds_2, y_train_2))
print('RMSE in train:', np.sqrt(mean_squared_error(train_preds_2, y_train_2)))
print('MAE in test:', mean_absolute_error(test_preds_2, y_test_2))
print('RMSE in test:', np.sqrt(mean_squared_error(test_preds_2, y_test_2)))
```

MAE in train: 0.08306322489944155  
RMSE in train: 0.31992940604542647  
MAE in test: 0.09609292502639916  
RMSE in test: 0.34148036538825066

In [62]:

```
# Precisión sobre train y test
score_train = svm.score(x_train, y_train_2)
print('El grado de precisión del modelo para el conjunto train es {}'.format(round(score_train, 4)))
score_test = svm.score(x_test, y_test_2)
print('El grado de precisión del modelo para el conjunto test es {}'.format(round(score_test, 4)))
```

El grado de precisión del modelo para el conjunto train es 0.9248.  
El grado de precisión del modelo para el conjunto test es 0.9107.

In [64]:

```
# Precisión, recall y f1 del modelo en conjunto train y test

accuracy_train = accuracy_score(y_train_2, train_preds_2, normalize = True)
accuracy_test = accuracy_score(y_test_2, test_preds_2, normalize = True)
print('Accuracy train: {}, Accuracy test: {}'.format(round(accuracy_train, 4), round(accuracy_test, 4)))

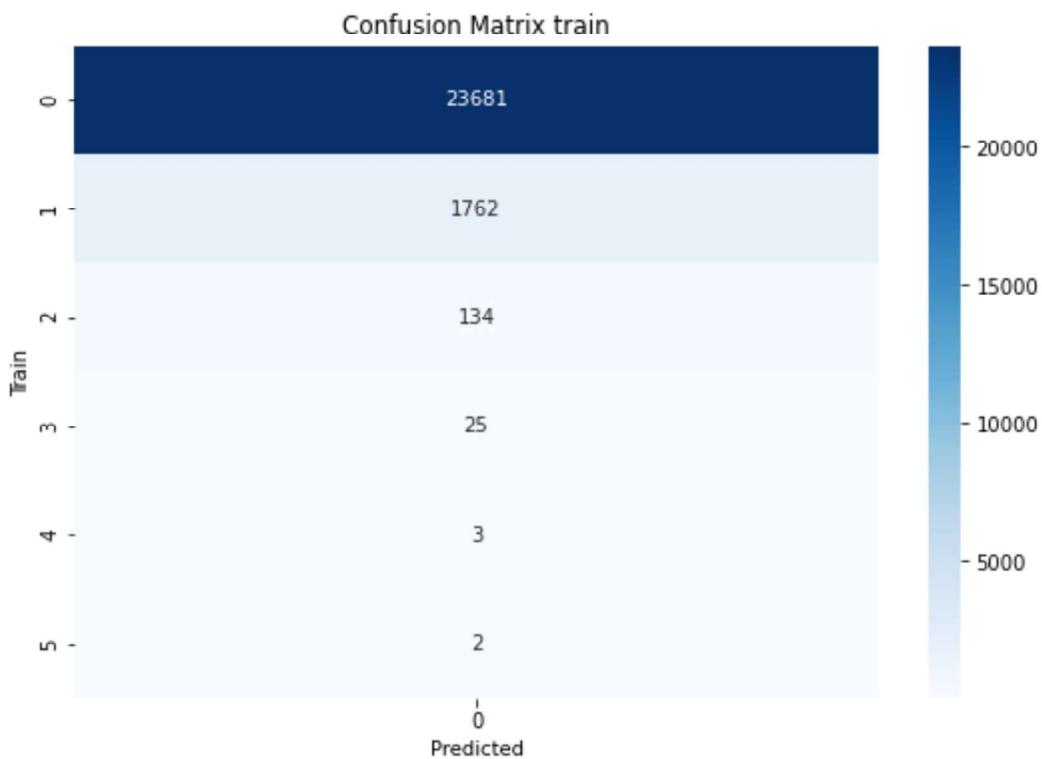
recall_train = recall_score(y_train_2, train_preds_2, average= 'weighted')
recall_test = recall_score(y_test_2, test_preds_2, average= 'weighted')
print('Recall train: {}, Recall test: {}'.format(round(recall_train, 4), round(recall_test, 4)))

f1_train = f1_score(y_train_2, train_preds_2, average= 'weighted')
f1_test = f1_score(y_test_2, test_preds_2, average= 'weighted')
print('F1 train: {}, F1 test: {}'.format(round(f1_train, 4), round(f1_test, 4)))
```

Accuracy train: 0.9248, Accuracy test: 0.9107  
Recall train: 0.9248, Recall test: 0.9107  
F1 train: 0.8886, F1 test: 0.8681

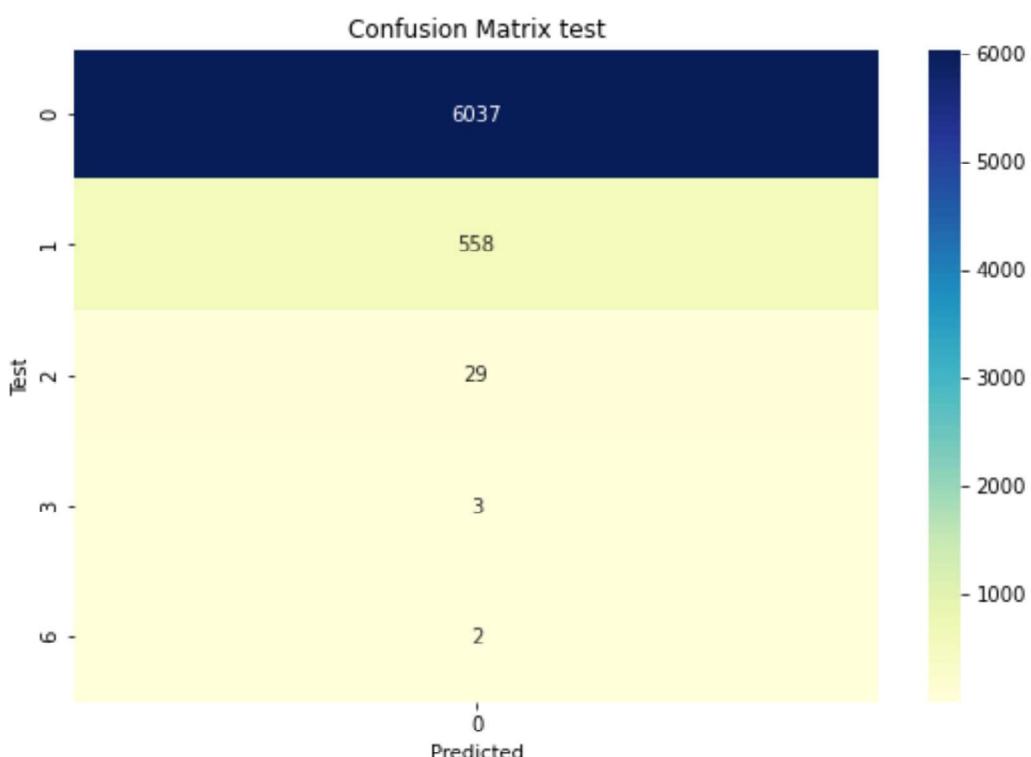
In [65]:

```
# Representación gráfica de la matriz de confusión sobre el conjunto train.
confusion_matrix_table = pd.crosstab(y_train_2, train_preds_2, rownames=['Train'])
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="Blues", fmt='g')
plt.title("Confusion Matrix train")
plt.savefig('graph/matriz_confusion_SVCpolytrain_2.jpg')
plt.show()
```



In [66]:

```
# Representación gráfica de la matriz de confusión sobre el conjunto test.
confusion_matrix_table = pd.crosstab(y_test_2, test_preds_2, rownames=['Test'], colnames=['Predicted'])
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="YlGnBu", fmt='g')
plt.title("Confusion Matrix test")
plt.savefig('graph/matriz_confusion_SVCpolytest_2.jpg')
plt.show()
```



### 3.4.2.3.2 Variable udsVenta discretizada en 14 intervalos de igual tamaño.

Se entrena en este ejemplo, la variable *udsVenta* discretizada en 14 intervalos de igual tamaño.

In [67]:

```
svm2 = SVC(kernel='poly', C=2, cache_size=100)

# Entrenamiento
svm2.fit(x_train, y_train_4)

# Predicción del modelo
train_preds_4 = svm2.predict(x_train)
test_preds_4 = svm2.predict(x_test)

#Vemos el error del algoritmo a la hora de predecir, para comprobar si a ajustado
print('MAE in train:', mean_absolute_error(train_preds_4, y_train_4))
print('RMSE in train:', np.sqrt(mean_squared_error(train_preds_4, y_train_4)))
print('MAE in test:', mean_absolute_error(test_preds_4, y_test_4))
print('RMSE in test:', np.sqrt(mean_squared_error(test_preds_4, y_test_4)))
```

```
MAE in train: 0.4102393876674347
RMSE in train: 0.8527966209922982
MAE in test: 0.4843867853371549
RMSE in test: 0.9172259067463356
```

In [68]:

```
# Precisión sobre train y test
score_train = svm2.score(x_train, y_train_4)
print('El grado de precisión del modelo para el conjunto train es {}'.format(round(score_train, 4)))
score_test = svm2.score(x_test, y_test_4)
print('El grado de precisión del modelo para el conjunto test es {}'.format(round(score_test, 4)))
```

```
El grado de precisión del modelo para el conjunto train es 0.6965.
El grado de precisión del modelo para el conjunto test es 0.6367.
```

In [70]:

```
# Precisión, recall y f1 del modelo en conjunto train y test

accuracy_train = accuracy_score(y_train_4, train_preds_4, normalize = True)
accuracy_test = accuracy_score(y_test_4, test_preds_4, normalize = True)
print('Accuracy train: {}, Accuracy test: {}'.format(round(accuracy_train, 4), round(accuracy_test, 4)))

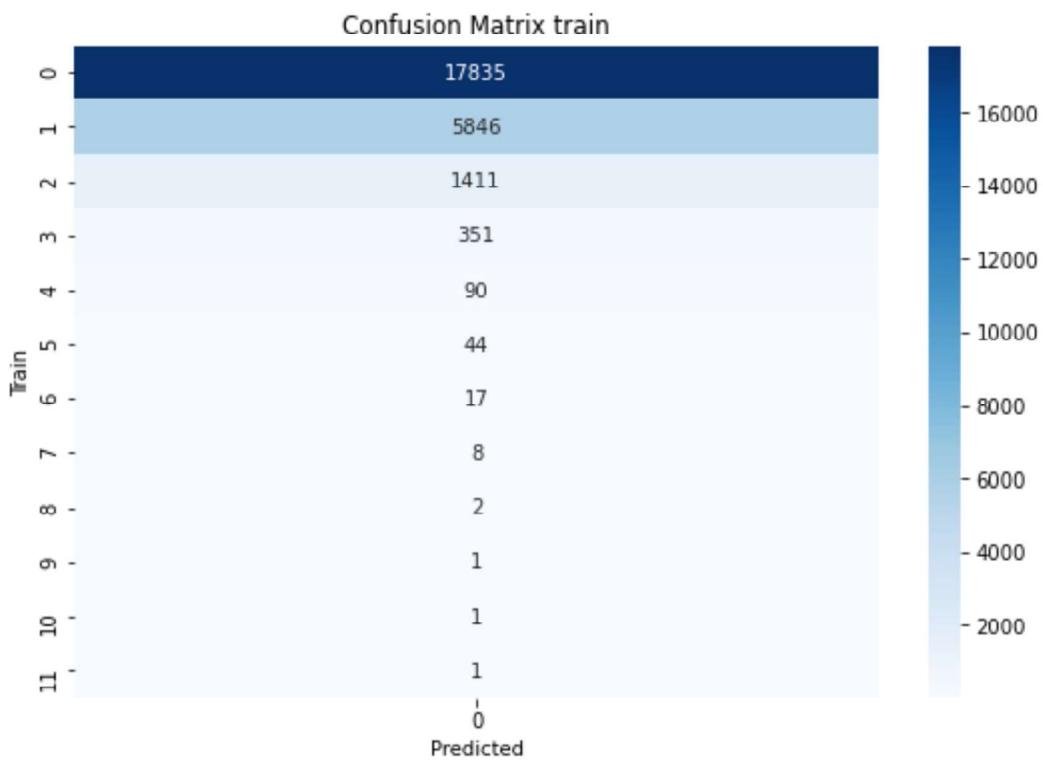
recall_train = recall_score(y_train_4, train_preds_4, average= 'weighted')
recall_test = recall_score(y_test_4, test_preds_4, average= 'weighted')
print('Recall train: {}, Recall test: {}'.format(round(recall_train, 4), round(recall_test, 4)))

f1_train = f1_score(y_train_4, train_preds_4, average= 'weighted')
f1_test = f1_score(y_test_4, test_preds_4, average= 'weighted')
print('F1 train: {}, F1 test: {}'.format(round(f1_train, 4), round(f1_test, 4)))
```

```
Accuracy train: 0.6965, Accuracy test: 0.6367
Recall train: 0.6965, Recall test: 0.6367
F1 train: 0.5719, F1 test: 0.4954
```

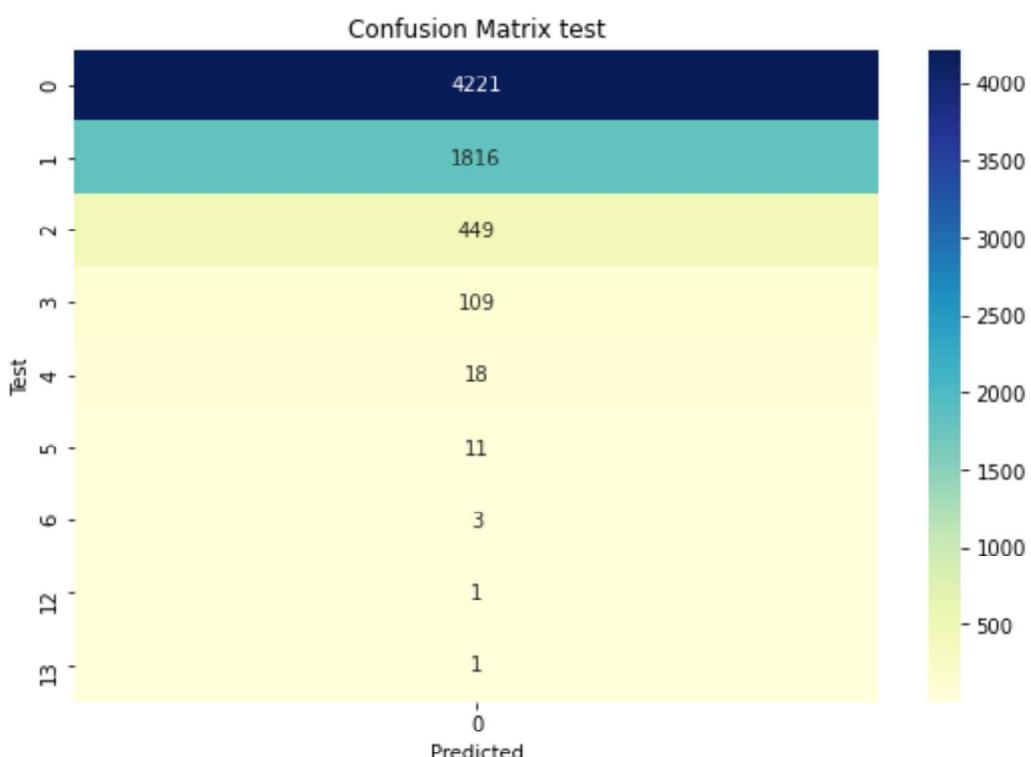
In [71]:

```
# Representación gráfica de la matriz de confusión sobre el conjunto train.
confusion_matrix_table = pd.crosstab(y_train_4, train_preds_4, rownames=['Train'])
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="Blues", fmt='g')
plt.title("Confusion Matrix train")
plt.savefig('graph/matriz_confusion_SVCpolytrain_2.jpg')
plt.show()
```



In [72]:

```
# Representación gráfica de la matriz de confusión sobre el conjunto test.
confusion_matrix_table = pd.crosstab(y_test_4, test_preds_4, rownames=['Test'], colnames=['Predicted'])
plt.figure(1, figsize=(9, 6))
sns.heatmap(confusion_matrix_table, annot=True, cmap="YlGnBu", fmt='g')
plt.title("Confusion Matrix test")
plt.savefig('graph/matriz_confusion_SVCpolytest_2.jpg')
plt.show()
```



### 3.4.2.3.3 Variable udsVenta discretizada en 25 clases, según valores únicos.

En este apartado se aborda algoritmo SVM con la variable *udsVenta* discretizada en 25 clases, según posibles valores únicos.

In [73]:

```
svm3 = SVC(kernel='poly', C=2, cache_size=100)

# Entrenamiento
svm3.fit(x_train, y_train_3)

# Predicción del modelo
train_preds_3 = svm3.predict(x_train)
test_preds_3 = svm3.predict(x_test)

#Vemos el error del algoritmo a La hora de predecir, para comprobar si a ajustado
print('MAE in train:', mean_absolute_error(train_preds_3, y_train_3))
print('RMSE in train:', np.sqrt(mean_squared_error(train_preds_3, y_train_3)))
print('MAE in test:', mean_absolute_error(test_preds_3, y_test_3))
print('RMSE in test:', np.sqrt(mean_squared_error(test_preds_3, y_test_3)))
```

MAE in train: 1.9352130276877417  
RMSE in train: 3.0404582385918286  
MAE in test: 2.1973148287826216  
RMSE in test: 3.231547987969824

In [74]:

```
# Precisión sobre train y test
score_train = svm3.score(x_train, y_train_3)
print('El grado de precisión del modelo para el conjunto train es {}'.format(round(score_train, 4)))
score_test = svm3.score(x_test, y_test_3)
print('El grado de precisión del modelo para el conjunto test es {}'.format(round(score_test, 4)))
```

El grado de precisión del modelo para el conjunto train es 0.3562.  
El grado de precisión del modelo para el conjunto test es 0.2978.

In [75]:

```
# Precisión, recall y f1 del modelo en conjunto train y test

accuracy_train = accuracy_score(y_train_3, train_preds_3, normalize = True)
accuracy_test = accuracy_score(y_test_3, test_preds_3, normalize = True)
print('Accuracy train: {}, Accuracy test: {}'.format(round(accuracy_train, 4), round(accuracy_test, 4)))

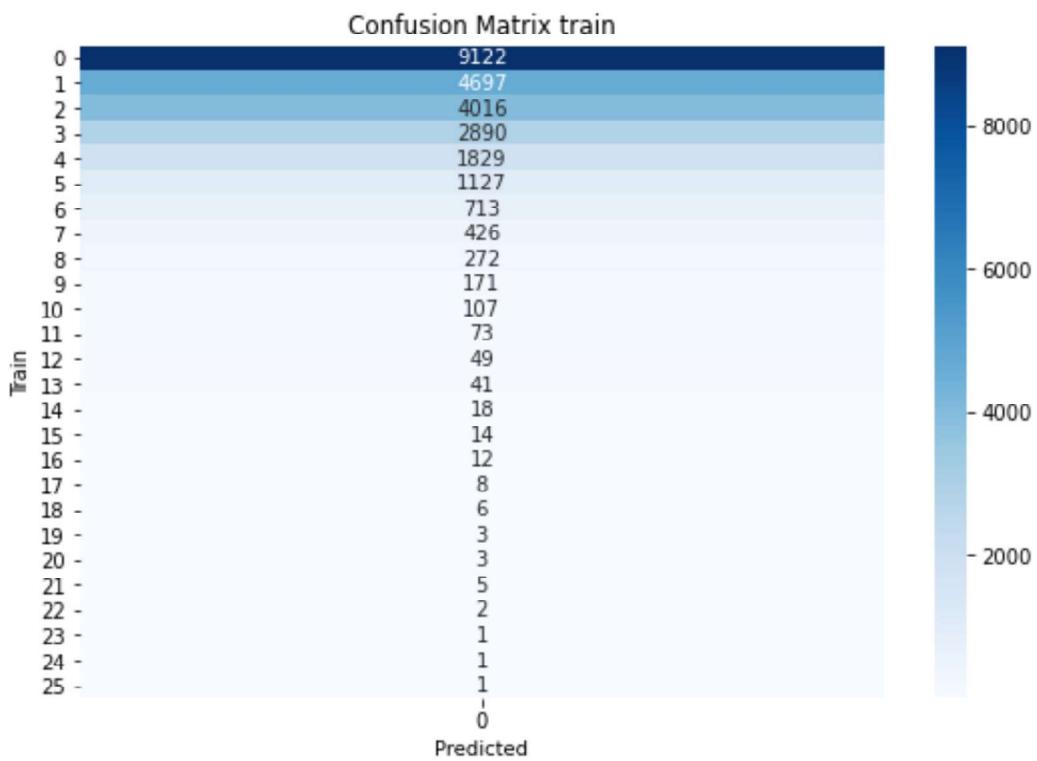
recall_train = recall_score(y_train_3, train_preds_3, average= 'weighted')
recall_test = recall_score(y_test_3, test_preds_3, average= 'weighted')
print('Recall train: {}, Recall test: {}'.format(round(recall_train, 4), round(recall_test, 4)))

f1_train = f1_score(y_train_3, train_preds_3, average= 'weighted')
f1_test = f1_score(y_test_3, test_preds_3, average= 'weighted')
print('F1 train: {}, F1 test: {}'.format(round(f1_train, 4), round(f1_test, 4)))
```

Accuracy train: 0.3562, Accuracy test: 0.2978  
Recall train: 0.3562, Recall test: 0.2978  
F1 train: 0.1871, F1 test: 0.1367

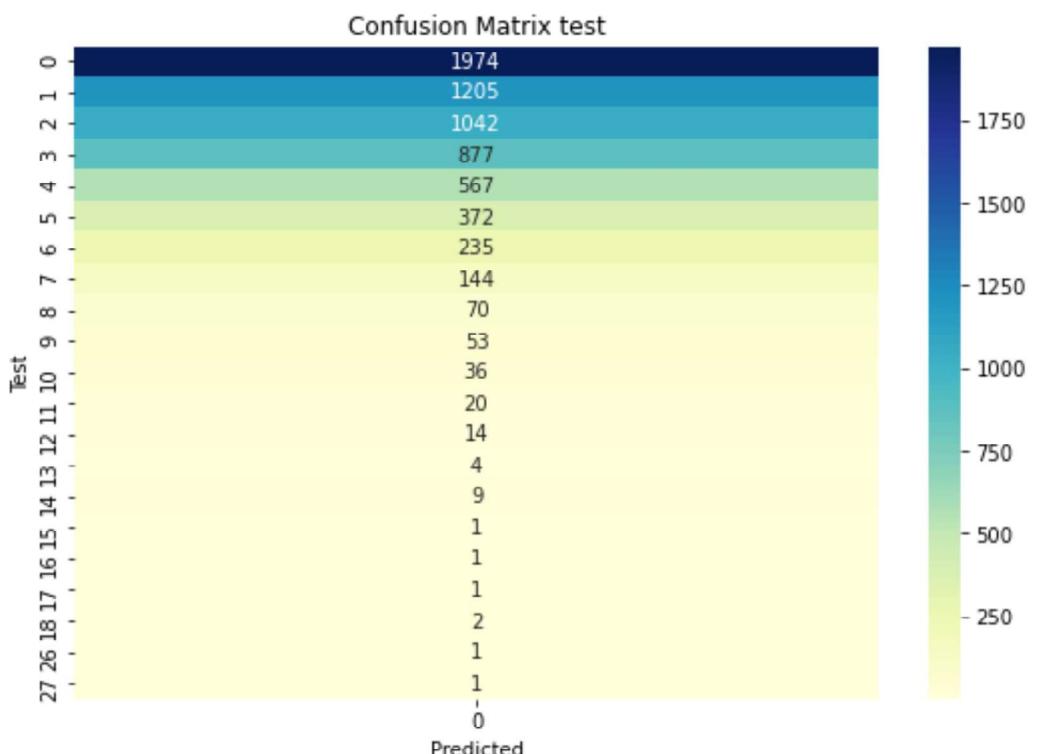
In [76]:

```
# Representación gráfica de la matriz de confusión sobre el conjunto train.
confusion_matrix_table = pd.crosstab(y_train_3, train_preds_3, rownames=['Train'])
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="Blues", fmt='g')
plt.title("Confusion Matrix train")
plt.savefig('graph/matriz_confusion_SVCpolytrain_3.jpg')
plt.show()
```



In [77]:

```
# Representación gráfica de la matriz de confusión sobre el conjunto test.  
confusion_matrix_table = pd.crosstab(y_test_3, test_preds_3, rownames=['Test'], colnames=['Predicted'])  
plt.figure(1, figsize=(9, 6))  
sns.heatmap(confusion_matrix_table, annot=True, cmap="YlGnBu", fmt='g')  
plt.title("Confusion Matrix test")  
plt.savefig('graph/matriz_confusion_SVCpolytest_3.jpg')  
plt.show()
```



Se guardan las predicciones en fichero

In [ ]:

```
# Predicciones de los diferentes modelos para el conjunto test.

test_preds_NB = test_preds_NB.astype(int)
test_preds_NB2 = test_preds_NB2.astype(int)
test_preds_SVM4 = test_preds_4.astype(int)
test_preds_SVM3 = test_preds_3.astype(int)
test_preds_SVM2 = test_preds_2.astype(int)
test_preds_SVM1 = test_preds_class.astype(int)

# Unión de todas las predicciones en un único fichero.
y_predict = np.column_stack((test_preds_NB, test_preds_NB2, test_preds_SVM4,
                             test_preds_SVM3, test_preds_SVM2, test_preds_SVM1))

# Transformación en dataframe
y_predict = pd.DataFrame(y_predict, columns=['test_preds_NB', 'test_preds_NB2', 'test_preds_SVM4',
                                               'test_preds_SVM3', 'test_preds_SVM2', 'test_preds_SVM1'])
```

In [79]:

```
y_predict.to_excel('data/y_predictsvm.xls')
```

### 3.4.2.4 Algoritmo SVC para variable categórica en días de venta.

La variable dependiente *udsVenta* tiene un elevado sesgo a la derecha, concentrando el 33% de las observaciones en el valor *udsVenta* = 0. Esto motiva que el modelo no sea eficaz en la predicción, asignando a la mayoría de las predicciones este valor. Se prueba de nuevo el algoritmo, pero prescindiendo de las observaciones en las que el establecimiento está cerrado (*\_bolOpen* = 0), en las que el 100% de *udsVenta* = 0. Se han eliminado 4.177 observaciones y *udsVenta* = 0 representa ahora el 23,35% de las observaciones.

Se ha suavizado el sesgo en la distribución de frecuencias.

La variable *udsVenta* se ha categorizado por su valor, definiéndose un total de 25 clases.

In [ ]:

```
# Modelo clasificación para variable dependiente discreta en días abiertos
# SVC (versión del modelo SVM)
# Hiperplano Lineal

svm = SVC(kernel='poly', C=2)

# Entrenamiento
svm.fit(x_train2, y_train2_3)

# Predicción del modelo
train_preds_5 = svm.predict(x_train2)
test_preds_5 = svm.predict(x_test2)

#Vemos el error del algoritmo a la hora de predecir, para comprobar si a ajustado
print('MAE in train:', mean_absolute_error(train_preds_5, y_train2_3))
print('RMSE in train:', np.sqrt(mean_squared_error(train_preds_5, y_train2_3)))
print('MAE in test:', mean_absolute_error(test_preds_5, y_test2_3))
print('RMSE in test:', np.sqrt(mean_squared_error(test_preds_5, y_test2_3)))
```

```
In [ ]: # Precisión, recall y f1 del modelo en conjunto train y test

accuracy_train = accuracy_score(y_train2_3, train_preds_5, normalize = True)
accuracy_test = accuracy_score(y_test2_3, test_preds_5, normalize = True)
print('Accuracy train: {}, Accuracy test: {}'.format(round(accuracy_train, 4), round(accuracy_test, 4)))

recall_train = recall_score(y_train2_3, train_preds_5, average= 'weighted')
recall_test = recall_score(y_test2_3, test_preds_5, average= 'weighted')
print('Recall train: {}, Recall test: {}'.format(round(recall_train, 4), round(recall_test, 4)))

f1_train = f1_score(y_train2_3, train_preds_5, average= 'weighted')
f1_test = f1_score(y_test2_3, test_preds_5, average= 'weighted')
print('F1 train: {}, F1 test: {}'.format(round(f1_train, 4), round(f1_test, 4)))
```

```
In [ ]: # Matriz de confusión
# confusion_matrix(y_test2_3, test_preds_5)
```

```
In [ ]: # Representación gráfica de la matriz de confusión sobre el conjunto train.

confusion_matrix_table = pd.crosstab(y_train2_3, train_preds, rownames=['Train'],
                                     colnames=['Predicted'])
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="Blues", fmt='g')
plt.title("Confusion Matrix train")
plt.savefig('graph/matriz_confusion_SVCtrain2_3.jpg')
plt.show()
```

```
In [ ]: # Representación gráfica de la matriz de confusión sobre el conjunto test.

confusion_matrix_table = pd.crosstab(y_test2_3, test_preds, rownames=['Test'],
                                      colnames=['Predicted'])
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="YlGnBu", fmt='g')
plt.title("Confusion Matrix test")
plt.savefig('graph/matriz_confusion_SVCtest2_3.jpg')
plt.show()
```

```
In [ ]: y_predict = pd.read_excel('data/y_predictsvm.xls')
test_preds_SVM5 = test_preds_5.astype(int)
y_predict['test_preds_Tree_4'] = test_preds_SVM5
y_predict.to_excel('data/y_predictsvm.xls')
```

### 3.4.3 Modelo Arbol de decisión

Arbol de decisión es otro algoritmo de ML para clasificación. Se prueba en este apartado la precisión del algoritmo aplicado a diferentes discretizaciones de la variable udsVenta.

#### 3.4.3.1 Árbol de decisión variable *udsVenta* categorizada.

Se aplica algoritmo para fichero completo con la variable *udsVenta* categorizada por clases.

```
In [38]: from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
y_train_class = label_encoder.fit_transform(y_train_1)
y_test_class = label_encoder.fit_transform(y_test_1)
```