Practica 3: k-vecinos más cercanos

3 de mayo de 2022

Dataset disponible en:
https://drive.google.com/file/d/19WVDYOM1xbF2hvbo75MDGe7OMN1clGhK/view?usp=sharing

1. Objetivo de la practica.

Usar un dataset de Pozos de Michoacán para hacer un modelo de KNN que agrupe elementos conforme al volumen de extracción de pozos en Michoacán (supervisado).

2. Conceptos.

Ya se habló en las prácticas anteriores de Pandas, scikitlearn, y Matplotlib así que toca hablar de las demás librerías que se usarán.
Seaborn es una biblioteca de visualización de datos de Python basada en matplotlib. Proporciona una interfaz de alto nivel para dibujar gráficos estadísticos atractivos e informativos.
El dataset consiste en un archivo .xlsx que contiene datos de de diferentes pozos acuiferos de Michoacán. Los datos incluyen sus coordenadas, municipio donde se encuentra, volumen (m^2/año), etc.

3. Herramientas a usar.

Computadora con acceso a internet.
Los siguientes programas y librerías:

a. Python versión 3.8.8

b. Anaconda versión 4.10.1

c. Jupyter-lab 3.0.14

d. Pandas versión 1.2.4

e. Scikitlearn versión 0.24.1

f. Matplotlib versión 3.3.4

g. Seaborn versión 0.11.1

h. openpyxl versión 3.0.7

i. Xlrd versión 2.0.1

4. Desarrollo.

4.1. Entender el problema.

Este problema consiste en tratar de clasificar los pozos en tipos de Uso, de acuerdo a una serie de características. Para lograrlo, se propone construir un modelo de clasificación basado en el algoritmo de K-vecinos más cercanos (KNN por sus siglas en inglés).

4.2. Criterio de evaluación.

Para esta práctica, se propone usar la precisión como métrica de evaluación.

4.3. Preparar los datos.

a. Importe las librerías necesarias.

```
import os
import path
import zipfile
import numpy as np
import pandas as pd
from sklearn.model_selection import GridSearchCV,
train_test_split, cross_val_score
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import plot_confusion_matrix,
precision_score, classification_report, recall_score, f1_score
import matplotlib.pyplot as plt
import seaborn as sns
```

b. Descargue los datos del link que se encuentra al inicio de esta práctica.

c. Extraer los datos.
Use el método mostrado en la práctica 0 sección 3 inciso h para extraerlos datos con Python OPCIONAL
Cree una carpeta para extraer ahí los archivos.

d. Cargar los datos.
IMPORTANTE
Tener instaladas las librerías *xlrd* y *openpyxl* para poder leer el archivo .xlsx
Importe la librería pandas y cargue el archivo que en este caso se llama "Pozos_Michoacan.xlsx"

```
read = pd.read_excel(Path+'Pozos_Michoacan.xlsx')
pozos = pd.DataFrame(pd.read_excel(Path+'Pozos_Michoacan.xlsx'))
```

Código 1

e. Explorar los datos.
Use head() para ver el contenido de los datos. También use info() para ver si existen registros vacíos. [Ver Figura 1]

```
pozos.head()
pozos.info()
```

	#	Título	Latitud	Longitud	Estado	Municipio	RegiónHidrológica	Cuenca	AcuíferoHomologado	Volumen (m3/año)	Uso
0	1	837156	19°58'36.0006"	-101°16'41.0007"	MICHOACÁN DE OCAMPO	HUANDACAREO	LERMA-SANTIAGO	188	1602 - MORELIA-QUERENDARO	22800.0	AGRICOLA
1	2	836988	19°59'22.0004"	-101°16'30.0003"	MICHOACÁN DE OCAMPO	HUANDACAREO	LERMA-SANTIAGO	188	1602 - MORELIA-QUERENDARO	9000.0	AGRICOLA
2	3	836992	20°02'22.0006"	-101°09'00.0005"	MICHOACÁN DE OCAMPO	CUITZEO	LERMA-SANTIAGO	188	1602 - MORELIA-QUERENDARO	20020.0	AGRICOLA
3	4	836822	20°01'50.0005"	-101°09'38.0006"	MICHOACÁN DE OCAMPO	CUITZEO	LERMA-SANTIAGO	188	1602 - MORELIA-QUERENDARO	6000.0	AGRICOLA
4	5	835739	19°49'11.0005"	-101°08'37.0006"	MICHOACÁN DE OCAMPO	TARÍMBARO	LERMA-SANTIAGO	188	1602 - MORELIA-QUERENDARO	30000.0	SERVICIOS

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1363 entries, 0 to 1362
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   #                   1363 non-null   int64
 1   Titulo              1363 non-null   object
 2   Latitud             1363 non-null   object
 3   Longitud            1363 non-null   object
 4   Estado              1363 non-null   object
 5   Municipio           1363 non-null   object
 6   RegiónHidrológica   1363 non-null   object
 7   Cuenca              1363 non-null   int64
 8   AcuiferoHomologado  1363 non-null   object
 9   Volumen (m3/año)    1363 non-null   float64
 10  Uso                 1363 non-null   object
dtypes: float64(1), int64(2), object(8)
memory usage: 117.3+ KB
```

Figura 1: Función head() y Función info()

Al igual que en la práctica 2, no hay datos nulos, pero se tienen que hacer modificaciones a los datos para que se pueda trabajar con ellos.

f. Preprocesamiento de los datos.
Primero se realiza una pequeña visualización de los datos para ver cual es el Uso más frecuente en el dataset. También una gráfica donde se muestre en dónde hay más pozos.

```
sns.displot(pozos['Uso'], palette="rocket")
sns.displot(pozos['Municipio'], palette="rocket")
```

Código 2

El resultado de estas instrucciones se pueden ver en la parte de abajo [Ver la Figura 2]

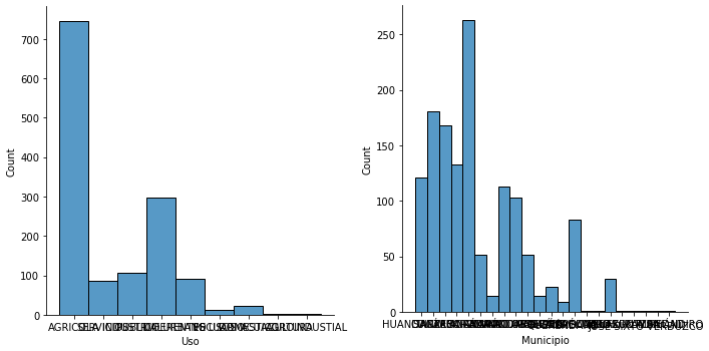


Figura 2: Usos y Municipios

Las etiquetas del eje x en la figura c son las siguientes (en orden):

- AGRÍCOLA
- PUBLICO URBANO
- INDUSTRIAL
- DIFERENTES USOS
- SERVICIOS
- DOMESTICO
- PECUARIO
- ACUACULTURA
- AGROINDUSTRIAL

En el caso de la figura d, las etiquetas son:

- MORELIA
- CUITZEO
- TARÍMBARO
- SANTA ANA MAYA
- HUANDACAREO
- COPÁNDARO
- ÁLVARO OBREGÓN
- ZINAPÉCUARO
- CHARO
- INDAPARAPEO
- HIDALGO
- CHUCÁNDIRO
- MORELOS
- ACUITZIO
- QUERÉNDARO
- ARIO
- LAGUNILLAS
- HUIRAMBA
- PURUÁNDIRO
- LA PIEDAD
- JOSÉ SIXTO VERDUZCO
- TZINTZUNTZAN

Primero se eliminan las columnas que no nos dan información. En este caso, las columnas "#", "Estado" y "Titulo" no nos brindan información útil ya que solo sirven para identificar el número de fila, por lo tanto se eliminan.
Use el siguiente código para eliminar las columnas.

```
pozos.drop(['#', 'Estado', 'Titulo'], axis = 1, inplace = True)
```

Código 3

Las columnas 'Latitud' y 'Longitud' son columnas con datos tipo *object*, por lo tanto es necesario cambiar el formato de los datos. Para esto, se propone tomar la coordenada y convertirla a segundos, Es decir, si se tiene la siguiente coordenada 20°01'50.0005", el procedimiento sería el siguiente:

$$(3600 * 20) + (60 * 01) + 50.0005 = 72110.0005$$

Use el siguiente código para aplicar el procedimiento anterior a todos los datos en las columnas 'Latitud' y 'Longitud'.

```
def grad2float(datos, id1,id2):
characters = '"'
lista = []
for i in datos:
    i = ''.join( x for x in i if x not in characters)
    if i[0] == '-':
        a = -1*(3600*float(i[1:id1]) + 60*float(i[id1:id2]) +
        float(i[id2:])) #Para longitud
    else:
        a = 3600*float(i[:id1]) + 60*float(i[id1:id2]) +
        float(i[id2:]) #Para latitud
    lista.append(float(a))
return lista

pozos['Latitud'] = grad2float(pozos['Latitud'], 2, 4)
pozos['Longitud'] = grad2float(pozos['Longitud'], 4, 6)
```

Código 4

En el caso de la columna Uso y Acuífero Homologado, podemos ver los siguientes datos usando la instrucción [Ver la Figura 3]

```
pozos['Uso'].value_counts()
pozos['AcuiferoHomologado'].value_counts()
```

```
AGRICOLA            745
PUBLICO URBANO      296
INDUSTRIAL          107
DIFERENTES USOS      91
SERVICIOS            87
DOMESTICO            23
PECUARIO             11
ACUACULTURA           2
AGROINDUSTIAL         1
Name: Uso, dtype: int64
```

```
1602  - MORELIA-QUERENDARO      1138
1602 - MORELIA-QUERENDARO        197
1610 - CIUDAD HIDALGO-TUXPAN      28
Name: AcuiferoHomologado, dtype: int64
```

Figura 3: Etiquetas Uso y Etiquetas Acuífero

Existen muy poquitos datos cuya etiqueta sea ACUACULTURA y AGROINDUSTIAL, así que se cambiarán esos datos a DIFERENTES USOS. En el caso del Acuífero Homologado hay dos etiquetas iguales, cuya única diferencia es un espacio extra, así que se deben unir esas etiquetas.
Inserte el siguiente código para cambiar las etiquetas mencionadas anteriormente.

```
pozos['Uso'].replace(['ACUACULTURA', 'AGROINDUSTIAL'],
'DIFERENTES USOS', inplace = True)
pozos['AcuiferoHomologado'].replace('1602  -  MORELIA-QUERENDARO',
'1602 - MORELIA-QUERENDARO', inplace = True)
```

El resultado de estos procedimientos se muestran a continuación [Ver Figura 4]

	Latitud	Longitud	Municipio	RegiónHidrológica	Cuenca	AcuiferoHomologado	Volumen (m3/año)	Uso
0	71916.0006	-364601.0007	HUANDACAREO	LERMA-SANTIAGO	188	1602 - MORELIA-QUERENDARO	22800.0	AGRICOLA
1	71962.0004	-364590.0003	HUANDACAREO	LERMA-SANTIAGO	188	1602 - MORELIA-QUERENDARO	9000.0	AGRICOLA
2	72142.0006	-364140.0005	CUITZEO	LERMA-SANTIAGO	188	1602 - MORELIA-QUERENDARO	20020.0	AGRICOLA
3	72110.0005	-364178.0006	CUITZEO	LERMA-SANTIAGO	188	1602 - MORELIA-QUERENDARO	6000.0	AGRICOLA
4	71351.0005	-364117.0006	TARÍMBARO	LERMA-SANTIAGO	188	1602 - MORELIA-QUERENDARO	30000.0	SERVICIOS

Figura 4: Datos

g. Dividir los datos en Train y test (X y y).

Use el siguiente código para dividir los datos en X y y. Estos ayudarán a entrenar el modelo.

```
X = pozos.iloc[:,:-1]
y = pozos['Uso']
```

Código 5

Compruebe que X y y tienen las columnas correctas usando head().[Ver la Figura 5]

```
X.head()
y.head()
```

	Latitud	Longitud	Municipio	RegiónHidrológica	Cuenca	AcuíferoHomologado	Volumen (m3/año)
0	71916.0006	-364601.0007	HUANDACAREO	LERMA-SANTIAGO	188	1602 - MORELIA-QUERENDARO	22800.0
1	71962.0004	-364590.0003	HUANDACAREO	LERMA-SANTIAGO	188	1602 - MORELIA-QUERENDARO	9000.0
2	72142.0006	-364140.0005	CUITZEO	LERMA-SANTIAGO	188	1602 - MORELIA-QUERENDARO	20020.0
3	72110.0005	-364178.0006	CUITZEO	LERMA-SANTIAGO	188	1602 - MORELIA-QUERENDARO	6000.0
4	71351.0005	-364117.0006	TARÍMBARO	LERMA-SANTIAGO	188	1602 - MORELIA-QUERENDARO	30000.0

```
0     AGRICOLA
1     AGRICOLA
2     AGRICOLA
3     AGRICOLA
4    SERVICIOS
Name: Uso, dtype: object
```

Figura 5: Datos en X y Datos en y

h. Codificar y Escalar los datos en X.
Como se tienen datos tipo *float* y *object* se van a codificar los datos en una misma función, la cual separará las columnas y les dará su respectiva codificación.

```
def codificar(data):
cat = []
num = []
for i in data.columns:
    if data[i].dtypes == 'object':
        cat.append(i)
    elif data[i].dtypes == 'float64' or 'int64':
        num.append(i)
scaler = MinMaxScaler()
scaled_values = scaler.fit_transform(data[num])
data[num] = scaled_values
code = LabelEncoder()
data[cat] = data[cat].apply(code.fit_transform)
print('num', num)
print('cat',cat)
return data

x_cod = X.copy()
X_cod = codificar(X)
```

Código 6

Use head() para verificar que los datos son tipo numérico. [Ver la Figura 6]

	Latitud	Longitud	Municipio	RegiónHidrológica	Cuenca	AcuíferoHomologado	Volumen (m3/año)
0	0.347490	0.735502	7	1	1.0	0	0.002066
1	0.351577	0.736699	7	1	1.0	0	0.000815
2	0.367570	0.785660	5	1	1.0	0	0.001814
3	0.364727	0.781525	5	1	1.0	0	0.000544
4	0.297290	0.788162	18	1	1.0	0	0.002718

Figura 6: X codificado

4.4. Construir el modelo.

Primero se deben dividir los datos en train y test. Use este código para dividir los datos.

```
X_train, X_test, y_train, y_test = train_test_split(X_cod, y,
test_size = 0.3, shuffle=True)
```

Código 7

Se construye un modelo simple de KNN con 3 vecinos (después se cambiará ese valor)

```
knn = KNeighborsClassifier(n_neighbors=3, weights='distance')
knn.fit(X_train, y_train)
```

Código 8

Una vez que el modelo termine de entrenar, se hacen las predicciones con ese modelo. Use este código para realizar predicciones y mostrar la métrica. [Ver la Figura 7]

```
def prediccion(model, X, y):
    pred = model.predict(X)
    y_pred = pd.Series(pred, index=y.index)
    print('Precision = ', precision_score(y, y_pred, average='micro'))
    print(classification_report(y, pred))
    plot_confusion_matrix(model, X, y)
    return y_pred

y_pred = prediccion(knn, X_test, y_test)
```

Código 9

Saber cuales fueron las etiquetas que sí acertó.

Como se tienen múltiples etiquetas, de las cuales hay algunas que se repiten 1 o 2 veces, es normal que el modelo no las logre identificar, ya que no tiene

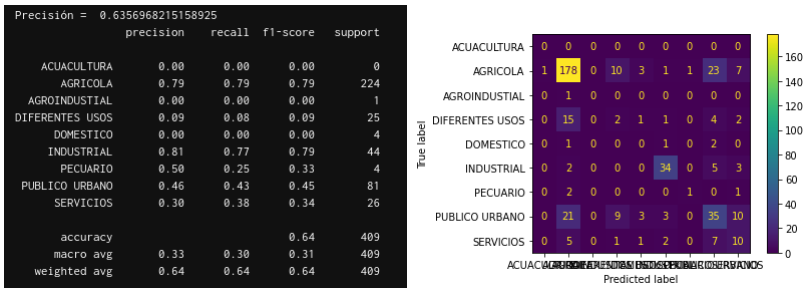


Figura 7: Métrica por etiqueta y Matriz de confusión

suficiente información para reconocerlas. Para contrarrestar esto, se propone identificar las etiquetas que el modelo sí acertó y usar el volumen para ponderar las predicciones de acuerdo al volumen de las etiquetas predichas correctamente.

Use este código para crear un diccionario con los índices y etiquetas de los elementos predichos correctamente.

```
def match(y_t, y_p):
    matched = {}
    aux = 0
    for i, j in zip(y_t, y_p):
        if i == j:
            matched[y_t.index[aux]]= i
        aux+=1
    return matched
aux = match(y_test, y_pred)
```

Código 10

Usar la lista de elementos para calcular volumen ponderado

Una vez que se tienen las etiquetas que fueron correctamente acertadas, estás se van a usar para obtener el volumen ponderado de cada una, con relación al volumen total.
La siguiente función toma el dataset pozos, la lista obtenida en el Código 10 y el volumen total. Regresa un DataFrame que contiene el índice de los datos, las etiquetas y su volumen poderado.

```
def volumen(data, indices, vol):
    vol_pond = {}
    for i in list(indices.keys()):
        v_pond = (data.iloc[i][6])/vol
        vol_pond[i] = indices[i], v_pond
    vol_pond = pd.DataFrame([[key, vol_pond[key][0],
```

```
    vol_pond[key][1]] for key in vol_pond.keys()],
    columns=['ID', 'Uso', 'Volumen Ponderado'])
    return vol_pond
volumen_total = sum(pozos['Volumen (m3/año)'].values)
volumen_pon = volumen(pozos, aux, volumen_total)

volumen_pon.groupby('Uso').sum()
```

Código 11

El resultado de ese código se muestra a continuación [Ver Figura 8]

Uso	ID	Volumen Ponderado
AGRICOLA	127330	0.103895
DIFERENTES USOS	3181	0.000815
DOMESTICO	1221	0.000668
INDUSTRIAL	19849	0.083953
PUBLICO URBANO	26740	0.027119
SERVICIOS	2728	0.001771

Figura 8: Volumen Ponderado

Estos resultados aún son un poco vagos, así que ahora se propone realizar el mismo análisis de volumen ponderado, pero por etiquetas. Es decir, sacar el porcentaje del volumen de cada etiqueta acertada con relación al volumen total de esa etiqueta.
La siguiente función funciona de manera muy similar a la mostrada en el Código 11 solo que en lugar de dividir entre un volumen total, divide entre el volumen total de la etiqueta con la que está trabajando en ese momento. También regresa un DataFrame con las mismas columnas.

```
def vol_lab(data, indices, vol):
    vol_pond = {}
    for i in list(indices.keys()):
        vl = vol[indices[i]] #volumen total por etiqueta
        v_pond = (data.iloc[i])/vl #Volumen ponderado por etiqueta
        vol_pond[i] = indices[i], v_pond
    vol_pond = pd.DataFrame([[key, vol_pond[key][0], vol_pond[key][1]] for key i
    return vol_pond
v = pozos.groupby('Uso').sum()
v = v['Volumen (m3/año)']
vol_label = vol_lab(pozos['Volumen (m3/año)'], aux, v)
```

```
vol_label.groupby('Uso').sum()
```

Código 12

El resultado se puede ver enseguida [Ver Figura 9]

Uso	ID	Volumen Ponderado
AGRICOLA	127330	0.240584
DIFERENTES USOS	3181	0.029509
DOMESTICO	1221	0.104795
INDUSTRIAL	19849	0.294933
PUBLICO URBANO	26740	0.122520
SERVICIOS	2728	0.065969

Figura 9: Volumen Ponderado por etiqueta

Si se suman los resultados ponderados, se tiene que el volumen acertado es de 0.85 es decir, que el modelo acertó al 85 % del volumen de pozos (de acuerdo a sus usos).
Este es un resultado favorable, pero se espera que modificando el modelo, este presente mejores resultados

4.5. Análisis de errores.

Como se puede ver en la sección anterior, la precisión es de 64 %. Los siguientes procedimientos son con la finalidad de ver si se puede mejorar esta métrica.

Use el siguiente código para crear un nuevo modelo de KNN y entrenarlo con validación cruzada. [Ver Figura 10]

```
knn_cv = KNeighborsClassifier()
cv_scores = cross_val_score(knn_cv, X_cod, y, cv=10)
print(cv_scores)
print("Score promedio", np.mean(cv_scores))
```

Código 13

Como se puede ver, la precisión mejora cuando $k = 5$. Para comprobar si el rendimiento del modelo **knn_cv** es mejor que el primer modelo, se usará el Código 12 para cuantificar el volumen ponderado por etiqueta que acierta.
Primero se crea un nuevo modelo con $k = 5$

```
[0.70072993 0.5620438  0.60583942 0.64705882 0.71323529 0.64705882
 0.67647059 0.66911765 0.63970588 0.52205882]
Score promedio 0.6383319021039073
```

Figura 10: Precisión de la validación cruzada

```
knn5 = KNeighborsClassifier(n_neighbors=5, weights='distance')
knn5.fit(X_train, y_train)
y_pred5 = prediccion(knn5, X_test, y_test)
```

El resultado de la predicción se encuentra en la Figura 11

```
Precisión =  0.6479217603911981
                precision    recall  f1-score   support

     ACUACULTURA       0.00      0.00      0.00         1
       AGRICOLA        0.76      0.82      0.79       226
DIFERENTES USOS        0.16      0.15      0.15        20
      DOMESTICO        0.25      0.17      0.20         6
     INDUSTRIAL        0.80      0.77      0.79        43
       PECUARIO        0.00      0.00      0.00         4
 PUBLICO URBANO        0.53      0.42      0.47        86
      SERVICIOS        0.24      0.30      0.27        23

       accuracy                           0.65       409
      macro avg        0.34      0.33      0.33       409
   weighted avg        0.64      0.65      0.64       409
```

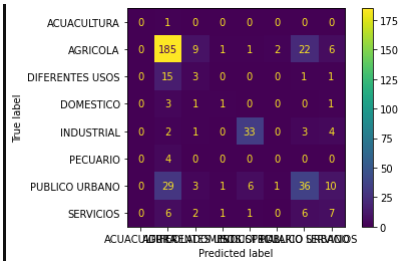


Figura 11: Precisión con k = 5 y Matriz con k = 5

Al usar la función del Código 10 y 12 se obtienen los siguientes resultados [Ver la Figura 12]

```
auxk5 = match(y_test, y_pred5)
vol_labelk5 = vol_lab(pozos['Volumen_(m3/año)'], auxk5, v)
vol_labelk5.groupby('Uso').sum()
```

Uso	ID	Volumen Ponderado
AGRICOLA	131908	0.245123
DIFERENTES USOS	2954	0.047044
DOMESTICO	1221	0.104795
INDUSTRIAL	20627	0.293822
PUBLICO URBANO	27568	0.126850
SERVICIOS	2540	0.053342

Figura 12: Volumen Ponderado por etiquetas con k = 5

Y la suma de estos volúmenes es 0.87, es decir que el modelo acertó al 87 %

del volumen de pozos. Esto es un 2 % mejor que el primer modelo.

Ahora se explorará el rendimiento de un modelo de knn generado por *GridSearchCV*.
Primero se genera un modelo knn sin parámetros y después se pone a entrenar usando valores de $k = [1, 25]$

```
knn2 = KNeighborsClassifier()
param_grid = {'n_neighbors': np.arange(1, 25)}
knn_gscv = GridSearchCV(knn2, param_grid, cv=10)
knn_gscv.fit(X_train, y_train)
```

Código 14

Si se usan las siguientes instrucciones se puede ver estos resultados [Ver la Figura 13]. Después obtenga el mejor estimador para ver el score en el test.

```
knn_gscv.best_params_
knn_gscv.best_score_
knn_grid = knn_gscv.best_estimator_ #Nuevo modelo
```

```
{'n_neighbors': 4}
```

```
0.675109649122807
```

Figura 13: K óptimo y Score

Use el Código 9 para realizar nuevas predicciones, cambiando el nombre de las variables y del modelo. [Ver la Figura 14]

```
y_pred_grid = prediccion(knn_grid, X_test, y_test)
```

```
Precisión =  0.6479217603911981
                precision    recall  f1-score   support

    ACUACULTURA       0.00      0.00      0.00         1
       AGRICOLA       0.72      0.86      0.79       226
DIFERENTES USOS       0.17      0.10      0.12        20
       DOMESTICO      0.50      0.17      0.25         6
      INDUSTRIAL      0.72      0.79      0.76        43
        PECUARIO      0.00      0.00      0.00         4
 PUBLICO URBANO       0.49      0.37      0.42        86
       SERVICIOS      0.09      0.04      0.06        23

        accuracy                           0.65       409
       macro avg       0.34      0.29      0.30       409
    weighted avg       0.60      0.65      0.62       409
```

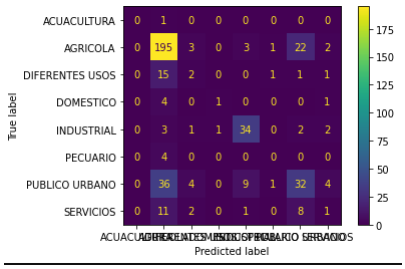


Figura 14: Precisión nueva y Matriz de confusión nueva

Ahora, con ayuda del Código 10 y Código 12 obtenga los volúmenes ponderados usando los resultados de este modelo. [Ver la Figura 15]

```
aux_grid = match(y_test, y_pred_grid)
vol_label_grid = vol_lab(pozos['Volumen (m3/año)'], aux_grid, v)
vol_label_grid.groupby('Uso').sum()
```

Uso	ID	Volumen Ponderado
AGRICOLA	139710	0.263035
DIFERENTES USOS	1949	0.030681
DOMESTICO	1221	0.104795
INDUSTRIAL	21520	0.294076
PUBLICO URBANO	25637	0.103706
SERVICIOS	158	0.006788

Figura 15: Volumen Ponderado por etiqueta

En este caso, la suma de estos datos es de 80 %. Significa que el modelo generado por grid tiene un peor rendimiento que el modelo generado or alidación cruzada. Debido a esto, se probará calcular k por fuerza bruta.

Use el siguiente código para graficar los scores de $k = [2, ..., 25]$ [Ver la Figura 16]

```
k_range = range(2, 25)
scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors = k)
    knn.fit(X_train, y_train)
    pred1=knn.predict(X_train)
    scores.append(precision_score(y_train, pred1, average='micro'))
#plt.figure()
plt.scatter(k_range, scores)
plt.xlabel('k')
plt.ylabel('Precision')
plt.grid()
plt.xticks([0,5,10,15,20])
plt.yticks([0, 0.25, 0.5, 0.75, 1])
```

Código 15

A partir de $k = 15$, la precisión empieza a oscilar, esto significa que en $k = 15$ se obtiene un buen score sin el sobreajuste. Haga un nuevo modelo con k = 15 y realice el mismo procedimiento que se hizo en los casos anteriores. [Ver Figura 17]

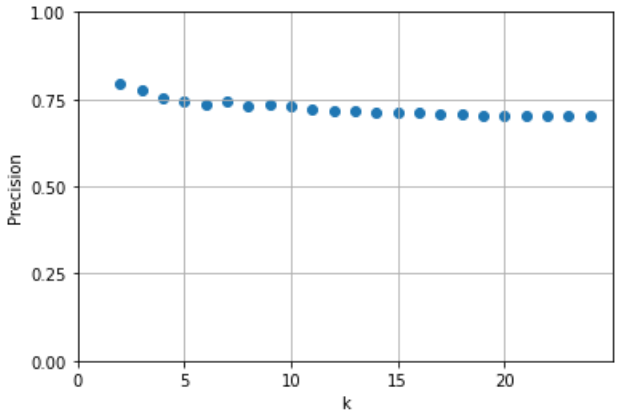
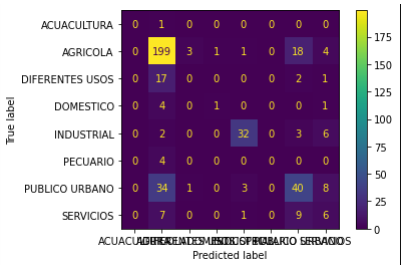


Figura 16: Scores de k

```
knn_k = KNeighborsClassifier(n_neighbors=15, weights='distance')
knn_k.fit(X_train, y_train)

y_pred_k = prediccion(knn_k, X_test, y_test)

auxk = match(y_test, y_pred_k)
vol_labelk = vol_lab(pozos['Volumen (m3/año)'], auxk, v)
vol_labelk.groupby('Uso').sum()
```

```
Precisión =  0.6797066014669927
                precision    recall  f1-score   support

    ACUACULTURA       0.00      0.00      0.00         1
       AGRICOLA       0.74      0.88      0.81       226
DIFERENTES USOS       0.00      0.00      0.00        20
      DOMESTICO       0.50      0.17      0.25         6
     INDUSTRIAL       0.86      0.74      0.80        43
       PECUARIO       0.00      0.00      0.00         4
PUBLICO URBANO        0.56      0.47      0.51        86
       SERVICIOS       0.23      0.26      0.24        23

       accuracy                           0.68       409
      macro avg       0.36      0.31      0.33       409
   weighted avg       0.64      0.68      0.65       409
```



Uso	ID	Volumen Ponderado
AGRICOLA	142787	0.253202
DOMESTICO	1221	0.104795
INDUSTRIAL	19654	0.293775
PUBLICO URBANO	29630	0.138795
SERVICIOS	2765	0.036870

Figura 17: Resultados con k = 15

La suma de estos resultados es 82 %. Así que el modelo que dio mejores resultados es el modelo generado por validación cruzada con *k=5*.

4.6. Implementación

Ver el Notebook de Jupyter llamado *P3-KNN.ipynb*