"ANEXO"

20 de septiembre de 2021

Practica 3: k-vecinos más cercanos

Dataset disponible en: https://drive.google.com/file/d/19WVDYOM1xbF2hvbo75MDGe7OMN1clGhK/vie

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

b. 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.

c. 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"

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

Código 1

d. Explorar los datos.
Use head() para ver el contenido de los datos. También use info() para ver si existen registros vacios. [Ver figuras a y b]

```
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

(a) Función head()

```
<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   Título              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   AcuíferoHomologado  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
```

(b) Función info

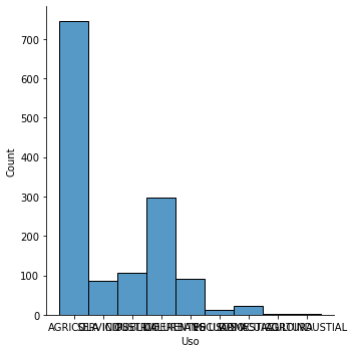
Al igual que en la práctica 2, no hay datos nulos, pero se tienen que hacer modificaciones a los datos para que se pueda trabajar con ellos.

e. 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.

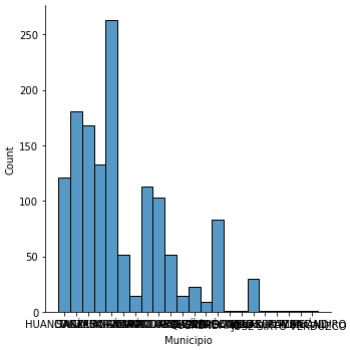
```
import seaborn as sns
sns.displot(pozos['Uso'], palette="rocket")
sns.displot(pozos['Municipio'], palette="rocket")
```

Código 2

El resultado de estas instrucciones se pueden ver en la parte de abajo [Ver las figuras c y d]



(c) Usos



(d) 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 las figuras e y f]

```
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
```

(e) Etiquetas Uso

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

(f) Etiquetas Acuifero

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 g]

	Latitud	Longitud	Municipio	RegiónHidrológica	Cuenca	AcuíferoHomologado	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

(g) Datos

f. Dividir los datos en 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 figuras h y i]

```
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

(h) Datos en X

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

(i) Datos en y

g. 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. Importe las librerías necesarias.

```
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
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 figura j]

	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

(j) X codificado

4.4. Construir el modelo.

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

```
from sklearn.model_selection import train_test_split
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)

```
from sklearn.neighbors import KNeighborsClassifier
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 figura k y l]

```
from sklearn.metrics import plot_confusion_matrix, precision_score, classificatio
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

Precisión = 0.6356968215158925

	precision	recall	f1-score	support
ACUACULTURA	0.00	0.00	0.00	0
AGRICOLA	0.79	0.79	0.79	224
AGROINDUSTIAL	0.00	0.00	0.00	1
DIFERENTES USOS	0.09	0.08	0.09	25
DOMESTICO	0.00	0.00	0.00	4
INDUSTRIAL	0.81	0.77	0.79	44
PECUARIO	0.50	0.25	0.33	4
PUBLICO URBANO	0.46	0.43	0.45	81
SERVICIOS	0.30	0.38	0.34	26
accuracy			0.64	409
macro avg	0.33	0.30	0.31	409
weighted avg	0.64	0.64	0.64	409

(k) Métrica por etiqueta

(l) Matriz de confusión

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 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 la figura m]

Uso	ID	Volumen Ponderado
AGRICOLA	126580	0.120718
DIFERENTES USOS	616	0.000169
INDUSTRIAL	20878	0.075899
PECUARIO	1008	0.000030
PUBLICO URBANO	22527	0.029778
SERVICIOS	3976	0.002158

(m) 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 n]

Uso	ID	Volumen Ponderado
AGRICOLA	126580	0.279542
DIFERENTES USOS	616	0.006126
INDUSTRIAL	20878	0.266637
PECUARIO	1008	0.032384
PUBLICO URBANO	22527	0.134535
SERVICIOS	3976	0.080380

(n) Volumen Ponderado por etiqueta

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 ñ]

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

Código 13

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

(ñ) Precisión de la validación cruzada

Como se puede ver, la precisión no ha mejorado. Ahora se usará GridSearchCV para ver si el score mejora.

```
from sklearn.model_selection import GridSearchCV
knn_2 = KNeighborsClassifier()
param_grid = {'n_neighbors': np.arange(1, 25)}
knn_gscv = GridSearchCV(knn_2, 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 figuras o y p]. 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': 21}
```

(o) K

```
0.6750657894736842
```

(p) Score

Use el código 9 para realizar nuevas predicciones, cambiando el nombre de las variables y del modelo. [Ver las figuras q y r]

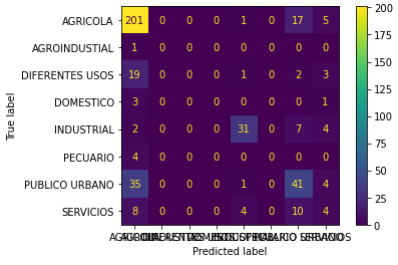
```
y_pred_grid = prediccion(knn_grid, X_test, y_test)
```

```
Precisión =  0.6772616136919315
                precision    recall  f1-score   support

      AGRICOLA       0.74      0.90      0.81       224
  AGROINDUSTIAL      0.00      0.00      0.00         1
DIFERENTES USOS      0.00      0.00      0.00        25
     DOMESTICO       0.00      0.00      0.00         4
    INDUSTRIAL       0.82      0.70      0.76        44
      PECUARIO       0.00      0.00      0.00         4
PUBLICO URBANO       0.53      0.51      0.52        81
     SERVICIOS       0.19      0.15      0.17        26

      accuracy                           0.68       409
     macro avg       0.28      0.28      0.28       409
  weighted avg       0.61      0.68      0.64       409
```

(q) Precisión nueva



(r) Matriz de confusión nueva

Ahora, con ayuda del Código 11 y Código 12 obtenga los volúmenes ponderados usando los resultados de este modelo. [Ver figuras s y t]

```
aux1 = match(y_test, y_pred_grid)
volumen_pon1 = volumen(pozos, aux1, volumen_total)
volumen_pon1.groupby('Uso').mean()
vol_label1 = vol_lab(pozos['Volumen_(m3/año)'], aux1, v)
vol_label1.groupby('Uso').mean()
```

Uso	ID	Volumen Ponderado
AGRICOLA	697.009950	0.000603
INDUSTRIAL	538.322581	0.002325
PUBLICO URBANO	630.804878	0.000782
SERVICIOS	481.250000	0.000210

(s) Volumen Ponderado

Uso	ID	Volumen Ponderado
AGRICOLA	697.009950	0.001397
INDUSTRIAL	538.322581	0.008166
PUBLICO URBANO	630.804878	0.003532
SERVICIOS	481.250000	0.007819

(t) Volumen Ponderado por etiqueta

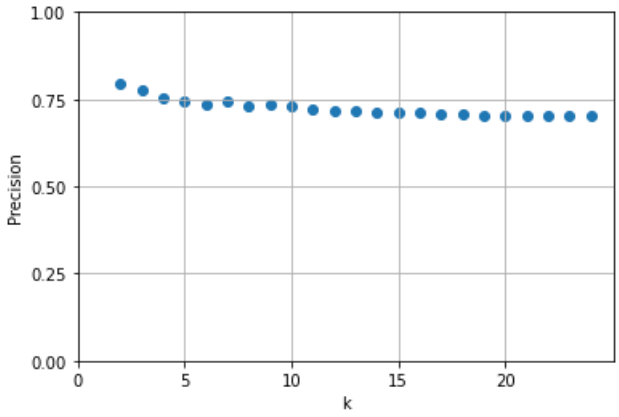
En este caso, ha identificado menos etiquetas que el modelo pasado, esto significa que el modelo tiene un sobreajuste debido al k tan grande. Debido a esto, se probará calcular k por fuerza bruta.

Use el siguiente código para graficar los scores de $k = [2, ..., 25]$ [Ver figura u]

```
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.



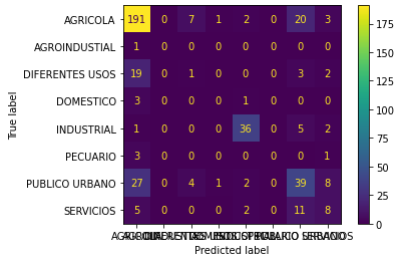
(u) Scores de k

Haga un nuevo modelo con k = 15 y realice el mismo procedimiento que se hizo en los casos anteriores. [Ver las figuras v-y]

Precisión = 0.6723716381418093

	precision	recall	f1-score	support
AGRICOLA	0.76	0.85	0.81	224
AGROINDUSTIAL	0.00	0.00	0.00	1
DIFERENTES USOS	0.08	0.04	0.05	25
DOMESTICO	0.00	0.00	0.00	4
INDUSTRIAL	0.84	0.82	0.83	44
PECUARIO	0.00	0.00	0.00	4
PUBLICO URBANO	0.50	0.48	0.49	81
SERVICIOS	0.33	0.31	0.32	26
accuracy			0.67	409
macro avg	0.31	0.31	0.31	409
weighted avg	0.63	0.67	0.65	409

(v) Precisión con k=15



(w) Matriz de confusión con k=15

Uso	ID	Volumen Ponderado
AGRICOLA	709.455497	0.000638
DIFERENTES USOS	417.000000	0.000075
INDUSTRIAL	621.777778	0.002115
PUBLICO URBANO	614.615385	0.000766
SERVICIOS	345.000000	0.000161

(x) Volumen Ponderado con k=15

Uso	ID	Volumen Ponderado
AGRICOLA	709.455497	0.001478
DIFERENTES USOS	417.000000	0.002717
INDUSTRIAL	621.777778	0.007430
PUBLICO URBANO	614.615385	0.003459
SERVICIOS	345.000000	0.005992

(y) Volumen Ponderado por etiqueta con k=15

En el caso de la precisión, se logró una mejora del 64 % al 67 %.

4.6. Implementación

Ver el Notebook de Jupyter llamado *P3-KNN_v2.ipynb*