UNIVERSIDAD
D
CÓRDOBA

INTRODUCCIÓN A LA MINERÍA DE DATOS

Práctica 1: Datos y exploración de datos

4º Curso - Grado en Ingeniería Informática

UCO-Escuela Politécnica Superior de Córdoba

Manuel Casas Castro - 31875931R

i72cascm@uco.es

ÍNDICE

Ejercicio 1 **3**

Ejercicio 2 **3**

Ejercicio 3 **5**

Ejercicio 4 **7**

Ejercicio 5 **10**

Ejercicio 6 **12**

Ejercicio 8 **13**

1. Obtenga 5 ejemplos de ficheros de datos en formato CSV, ARFF u otro cualquiera de:

◦ **Weka datasets**
◦ **UCI MLR**

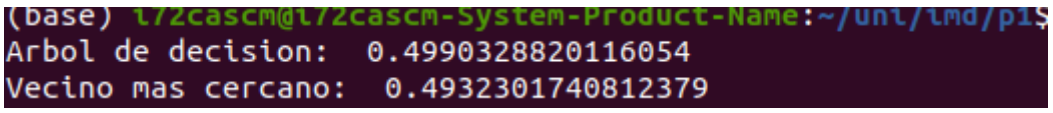
Los ficheros de datos obtenidos para la realización de esta práctica son:
-forestfires
-glass
-iris
-weather
-diabetes

Todos los ficheros han sido descargados y transformados a su formato “.csv”.

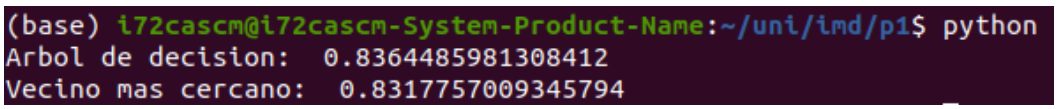
2. Evalúe el árbol de decisión y el vecino más cercano sobre los datos originales.

A continuación se muestran los resultados de aplicar árbol de decisión y vecino más cercano a los datos originales:

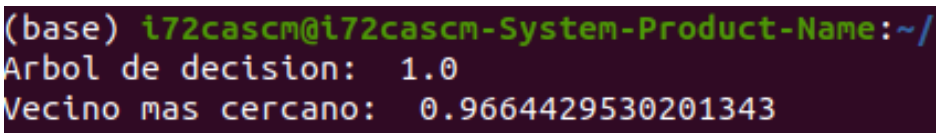
-forestfires:



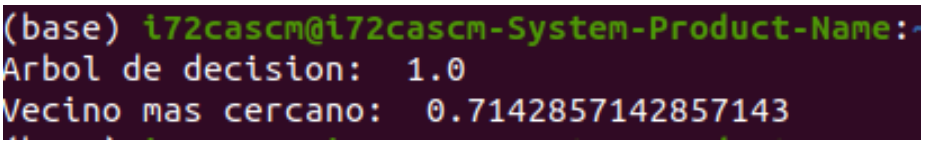
-glass:



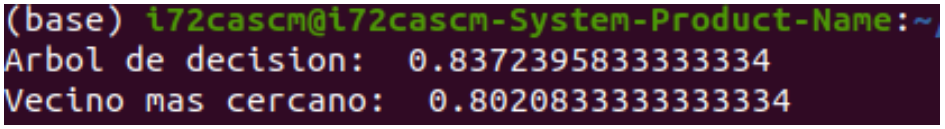
-iris:



-weather:



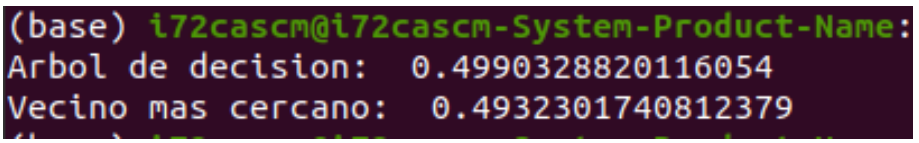
-diabetes:



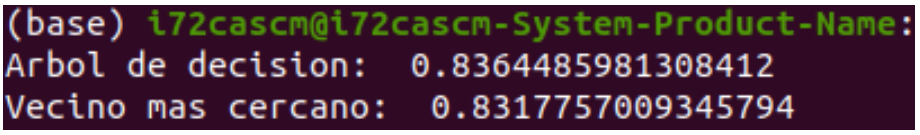
Podemos observar en ambos métodos que sus scores son muy parecidos para un mismo conjunto de datos, aunque de igual manera podemos observar que en todos los casos, el método “árbol de decisión” obtiene resultados levemente superiores al método “vecino más cercano”.

3. Estudie el efecto de la normalización (reescalar en el intervalo [0, 1]) y la estandarización (μ = 0, σ = 1) sobre el error de clasificación usando el árbol de decisión y el vecino más cercano. Comente los resultados.

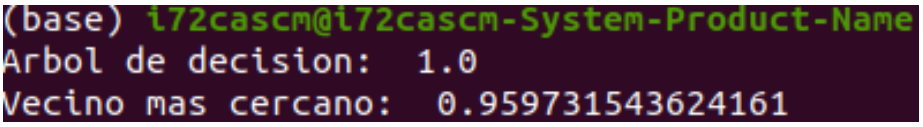
-forestfires:



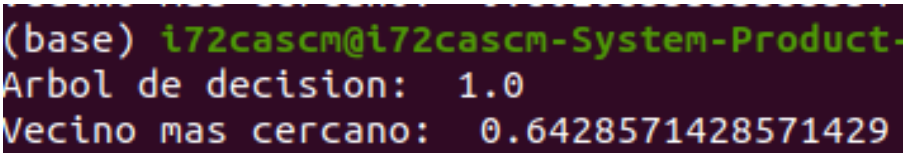
-glass:



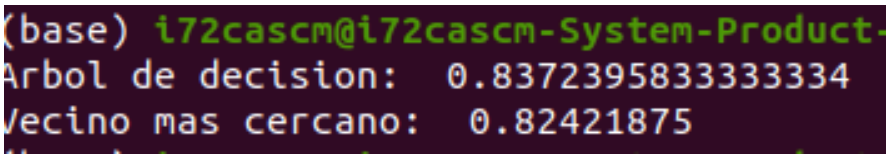
-iris:



-weather:



-diabetes:



Tras aplicar la normalización a los diferentes datasets, podemos observar que algunos resultados son similares a la ejecución del ejercicio anterior, como pueden ser forestfires.csv o glass.csv pero en los otros conjuntos de datos, el método de vecino más cercano varía con respecto al ejercicio anterior, en algunos casos hacia un peor resultado y en otros hacia uno mejor.
El método árbol de decisión se mantiene igual en todas las ejecuciones.

4. Estudie el efecto del análisis en componentes principales sobre el árbol de decisión y el vecino más cercano.

Para este ejercicio, seleccionaremos un solo datasets para realizar diversas pruebas sobre el mismo conjunto variando los parámetros de las funciones de entrenamiento.

-forestfires:

Utilizaremos en primer lugar los valores por defecto de las funciones de los clasificadores:

```
tree = DecisionTreeClass
    criterion = "gini",
    splitter = "best",
    max_depth = 5)

neighbors = KNeighborsCl
    n_neighbors = 5,
    weights = "uniform",
    algorithm = "auto")
```

```
(base) i72cascm@i72cascm-System-Product-Na
Score del árbol:  0.4951644100580271
Score del neighbors:  0.4932301740812379
```

Procedemos a cambiar los valores de los parámetros para analizar los resultados obtenidos:

```
tree = DecisionTreeClassifier(
    criterion = "gini",
    splitter = "best",
    max_depth = 10)

neighbors = KNeighborsClassifier(
    n_neighbors = 10,
    weights = "uniform",
    algorithm = "auto")
```

```
(base) i72cascm@i72cascm-System-Product-N
Score del árbol:  0.6054158607350096
Score del neighbors:  0.4796905222437137
```

Podemos observar para este experimento que al aumentar considerablemente la profundidad del árbol de decisión se obtienen mejores resultados, pero al aumentar el número de vecinos se obtienen peores resultados en ese clasificador.
En el caso de los árboles de decisión, puede deberse que al ser una base de datos con un número extenso de variables (13), ya que podría clasificar de una manera más eficiente, mientras que en los k-vecinos más cercanos puede estar empeorando ya que al subir el número de vecinos también aumentamos el sesgo de los mismos, lo que puede estar llevando a peores clasificaciones.

```
tree = DecisionTreeClassifier(
    criterion = "gini",
    splitter = "random",
    max_depth = 5)

neighbors = KNeighborsClassifier(
    n_neighbors = 5,
    weights = "distance",
    algorithm = "kd_tree")
```

```
(base) i72cascm@i72cascm-System-Product-Name
Score del árbol:  0.4951644100580271
Score del neighbors:  0.9825918762088974
```

En esta ejecución, en los árboles de decisión se utiliza un split aleatorio en vez del mejor split y por consecuencia, podemos observar que para un mismo número de profundidad del árbol el score del árbol es peor que cuando se elige el mejor split.

Por otro lado, si en el k-vecinos más cercanos utilizamos un sistema de pesos no uniforme donde vecinos más cercanos a un punto tienen más peso que los alejados. Podemos notar que este cambio incrementa enormemente la eficacia de este clasificador.

```
tree = DecisionTreeClassifier(
    criterion = "entropy",
    splitter = "best",
    max_depth = None)

neighbors = KNeighborsClassifier(
    n_neighbors = 3,
    weights = "distance",
    algorithm = "kd_tree")
```

```
(base) i72cascm@i72cascm-Syst
Arbol:  0.9825918762088974
Vecinos:  0.9825918762088974
```

Por último, en este experimento utilizaremos entropía y dejaremos que el clasificador decida la mejor profundidad para el árbol de decisión y en el caso de los vecinos utilizamos el método de distancia con menos vecinos para reducir el sesgo.
Podemos observar que son los mejores resultados obtenidos de los experimentos realizados.

5. Estudie el efecto del muestreo aleatorio del 10% de las instancias sin reemplazamiento sobre el árbol de decisión y el vecino más cercano. Comente los resultados. Compare los resultados con un muestreo igual estratificado.

Utilizaremos la función train_test_split para dividir el dataset en un conjunto de entrenamiento con el 90% de los datos y un conjunto de test con el 10% de los mismos de manera estratificada:

```
X_train, X_test, Y_train, Y_test = train_test_split(x, y, stratify=y, test_size = 0.1)
```

De esta manera, dividimos los datos en torno a la clase “y” para repartir los datos entre train y test de una manera más equitativa.

-glass

```
(base) i72cascm@i72cascm-System-Product-Nam
Arbol normal:  0.9047619047619048
Vecinos normal:  0.6190476190476191
Arbol estratificado:  0.8125
Vecino estratificado:  0.8072916666666666
```

-iris

```
(base) i72cascm@i72cascm-System-Product-Name:~
Arbol normal:  1.0
Vecinos normal:  0.8666666666666667
Arbol estratificado:  1.0
Vecino estratificado:  0.9701492537313433
```

-diabetes

```
(base) i72cascm@i72cascm-System-Product-Na
Arbol normal:  0.922077922077922
Vecinos normal:  0.7662337662337663
Arbol estratificado:  0.8335745296671491
Vecino estratificado:  0.8002894356005789
```

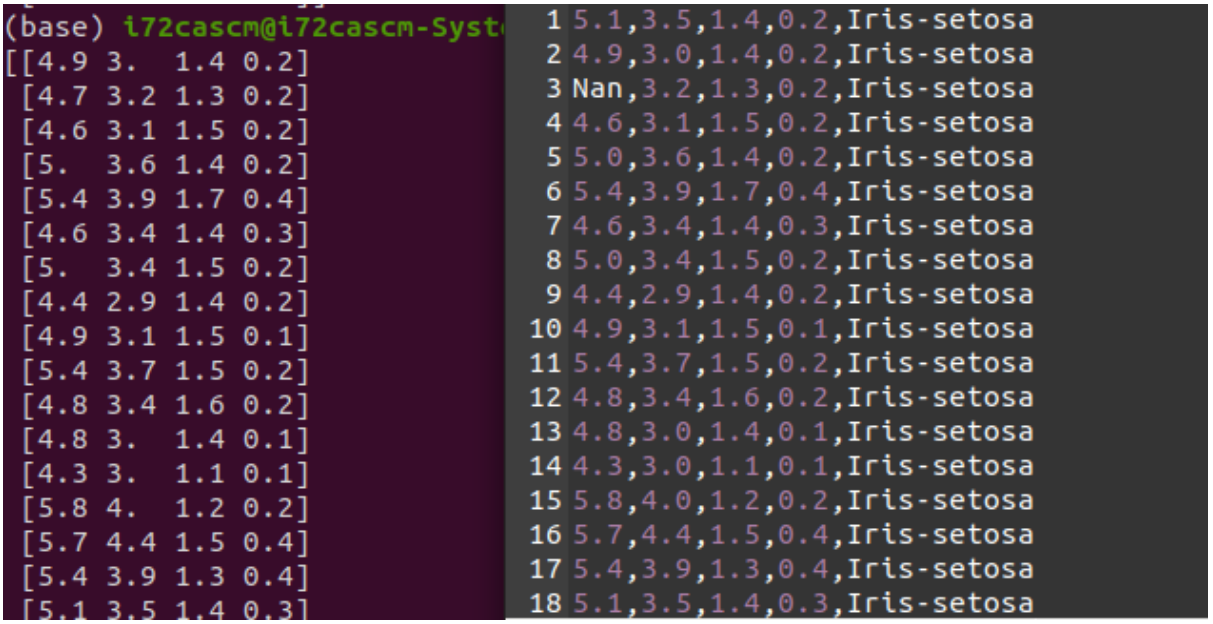
Podemos observar que para árboles la manera no estratificada no genera mejores resultados que la manera estratificada mientras que para clasificación mediante vecinos genera mejores resultados.

6. Valores perdidos. Existen diferentes métodos para imputar valores perdidos en la biblioteca scikit learn (https://scikit-learn.org/stable/modules/impute.html). Seleccione un conjunto de datos con valores perdidos y dos métodos de imputación. Estudie el efecto de los métodos de imputación sobre los dos clasificadores.

```
df = pd.read_csv("iris.csv")

X = df.iloc[:, :-1].values
Y = df.iloc[:, -1].values

imputer = KNNImputer(n_neighbors=2, weights="uniform")
print(imputer.fit_transform(X))
```



Al no estar utilizando ningún dataset con valores perdidos, se ha procedido a perder intencionadamente un valor del dataset iris.csv como ilustra la imagen. Utilizando las líneas de código mostradas en la imagen anterior, podemos estimar un valor para el valor perdido a partir de los datos del dataset para de esta forma obtener un dataset sin valores perdidos.

8. Discretización. Existen diferentes métodos de discretización en la biblioteca scikit learn https://scikit-learn.org/stable/modules/preprocessing.html#discretization). Seleccione un conjunto de datos y un método de discretización. Estudie el efecto del método sobre los dos clasificadores.

```
df = df.apply(preprocessing.LabelEncoder().fit_transf

x = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
est = preprocessing.KBinsDiscretizer().fit(x)
print(est.transform(x))
```

```
  (0, 0)        1.0
  (0, 7)        1.0
  (0, 10)       1.0
  (0, 16)       1.0
  (1, 0)        1.0
  (1, 8)        1.0
  (1, 10)       1.0
```

Siguiendo con el dataset iris.csv, aplicamos las siguientes líneas de código para generar intervalos en nuestro dataset.

Valores de utilizar los clasificadores con discretización:

```
  (148, 13)     1.0
  (148, 18)     1.0
Arbol de decision:  1.0
Vecino mas cercano:  0.9664429530201343
```

Para este caso podemos observar que no existen diferencias significativas respecto al no discretizado.