8 DE NOVIEMBRE DE 2019

PRÁCTICA 3: REDES NEURONALES DE BASE RADIAL

INTRODUCCIÓN A LOS MODELOS COMPUTACIONALES

ALBERTO LUQUE RIVAS
31018546X
I62LURIA@UCO.ES

GRADO EN INGENIERÍA INFORMÁTICA
COMPUTACIÓN
4º CURSO

UNIVERSIDAD D CÓRDOBA

ESCUELA POLITÉCNICA SUPERIOR DE CÓRDOBA
Universidad de Córdoba

EPSC

Contenido

1. Introducción

El trabajo que se va a realizar en la práctica consiste en implementar una red neuronal de tipo RBF realizando un entrenamiento en tres etapas:

1. Aplicación de un algoritmo de clustering que servirá para establecer los centros de las funciones RBF (pesos de capa de entrada a capa oculta).
2. Ajuste de los radios de las RBF, mediante una heurística simple (media de las distancias hacia el resto de los centros).
3. Aprendizaje de los pesos de capa oculta a capa de salida.
    a. Para problemas de regresión, utilización de la pseudo-inversa de Moore Penrose.
    b. Para problemas de clasificación, utilización de un modelo lineal de regresión logística.

2. Descripción de los pasos a realizar para llevar a cabo el entrenamiento de las redes RBF

Primero hemos hecho uso del algoritmo k medias para poder calcular los centroides en las mejores posiciones. Para dichos centroides se ha usado un pequeño Split del conjunto de entrenamiento cuando ha sido un problema de clasificación, cuando es de regresión los centros son aleatorios.

Seguidamente de calcular los centroides se calcula el algoritmo k medias de la librería usada, que básicamente lo que hace es posicionar de manera óptima los centroides para agrupar las clases de los dataset en el centroide mas adecuado el cual agrupara el mayor número de instancias.

Acto seguido con los centroides colocados en la posición optima para el problema se han calculado los radios de cada uno de ellos, esto se utiliza para saber a que clase pertenece cada patrón basándonos en la distancia al centroide más cercano.

Luego de tener los radios calculados se deben de ajustar los pesos de la capa de salida, la cual se ajustará dependiendo del tipo de problema, regresión o clasificación.

- **Regresión:** Pseudo-inversa por Moore-Penrose
- **Clasificación:** Se usará la regresión logística.

3. Experimentos y análisis de resultados

3.1 Breve descripción de las bases de datos utilizadas.

En los experimentos usados hemos tenido acceso a cuatro bases de datos cada una de las con su archivo de prueba y de train correspondiente al dataset. Las bases de datos usadas han sido:

- **Base de datos SIN:** esta base de datos está compuesta por 120 patrones de train y 41 patrones de prueba. Ha sido obtenida añadiendo cierto ruido aleatorio a la función seno.

- **Base de datos Parkinson:** esta base de datos está compuesta por 4406 patrones de train y 1469 patrones de prueba. Contiene, como entradas o variables independientes, una serie de datos clínicos de pacientes con la enfermedad de Parkinson y datos de medidas biométricas de la voz, y, como salidas o variables dependientes, el valor motor y total del UPDRS.

- **Base de datos Quake:** esta base de datos está compuesta por 1633 patrones de train y 546 patrones de prueba. Se corresponde con una base de datos en la que el objetivo es averiguar la fuerza de un terremoto (medida en escala sismológica de Richter). Como variables de entrada, utilizamos la profundidad focal, la latitud en la que se produce y la longitud.

- Base de datos vote: vote contiene 326 patrones de entrenamiento y 109 patrones de prueba. La base de datos incluye los votos para cada uno de los para cada uno de los candidatos para el Congreso de los EE. UU., identificados por la CQA. Todas las variables de entrada son categóricas.

- Base de datos noMNIST: esta base de datos, originariamente, está compuesta por 200000 patrones de entrenamiento y 10000 patrones de prueba, y un total de 10 clases. No obstante, para la práctica que nos ocupa, se ha reducido considerablemente el tamaño de la base de datos para realizar las pruebas en menor tiempo. Por lo tanto, la base de datos que sutilizara está compuesta por 900 patrones de entrenamiento y 300 patrones de prueba. Está formada por un conjunto de letras (de la a a la f) escritas con diferentes tipografías o simbologías. Están ajustadas a una rejilla cuadrada de 28 28 píxeles. Las imágenes están en escala de grises en el intervalo [□1;0; +1;0].9. Cada uno de los pixeles forman parte de las variables de entrada (con un total de 28 28 = 784 variables de entrada) y las clases se corresponden con la letra escrita (a, b, c, d, e y f, con un total de 6 clases).

Para cada uno de los datases mencionados previamente las pruebas realizadas han sido con un numero de nodos en cada una de las capas usadas han sido 5%,15%,25%,50%.

Para la prueba más fructífera de cada dataset se han realizado pruebas adicionales variando los factores de eta y de penalty.

3.2 Breve descripción de los valores de los parámetros considerados.

- Argumento `-t`, `--train_file`: Indica el nombre del fichero que contiene los datos de entrenamiento a utilizar. Sin este argumento, el programa no puede funcionar.
- Argumento `-T`, `--test_file`: Indica el nombre del fichero que contiene los datos de *test* a utilizar. Si no se especifica este argumento, utilizar los datos de entrenamiento como *test*.
- Argumento `-c`, `--classification`: Booleano que indica si el problema es de clasificación. Si no se especifica, supondremos que el problema es de regresión.
- Argumento `-r`, `--ratio_rbf`: Indica la razón (en tanto por uno) de neuronas RBF con respecto al total de patrones en entrenamiento. Si no se especifica, utilizar $0{,}1$ capa oculta.
- Argumento `-l`, `--l2`: Booleano que indica si utilizaremos regularización de L2 en lugar de la regularización L1. Si no se especifica, supondremos que regularización L1.
- Argumento `-e`, `--eta`: Indica el valor del parámetro *eta* (η). Por defecto, utilizar $\eta = 1e{-}2$.
- Argumento `-o`, `--outputs`: Indica el número de columnas de salida que tiene el conjunto de datos y que siempre están al final. Por defecto, utilizar $o = 1$.
- (Kaggle) Argumento `-p`, `--pred`: Booleano que indica si utilizaremos el modo de predicción.
- (Kaggle) Argumento `-m`, `--model_file`: Indica el directorio en el que se guardarán los modelos entrenados (en el modo de entrenamiento, sin el *flag* p) o el fichero que contiene el modelo que se utilizará (en el modo de predicción, con el *flag* p).
- Argumento `--help`: Mostrar la ayuda del programa (utilizar la que genera automáticamente la librería `click`).

3.3 Resultados obtenidos y análisis de estos.

Se van a mostrar los resultados de la experimentación de todas las pruebas realizadas, están bien diferenciadas en dos tipos, las pruebas realizas con los dataset de regresión y las pruebas realizadas con las pruebas de clasificación, aunque en última instancia usaremos un dataset de clasificación y lo lanzaremos como si se tratara de un problema de regresión. Para el dataset de nomnist se mostrará la matriz de confusión resultante y se interpretaran los datos de esta.

3.3.1 Dataset SIN

REGRESION KMEANS INIT RAMDOM

PORCENTAJE DE NODOS	5%	15%	25%	50%
MSE TRAIN	0,013798	0,011231	0,010352	0,11472
MSE TEST	0,02231	0,109643	0,124171	0,195356
CCR TRAIN	80,33	78,33	79,5	80,83
CCR TEST	78,05	65,85	67,8	69,27

REGRESION KMEANS INIT k-means++

PORCENTAJE DE NODOS	5%	15%	25%	50%
MSE TRAIN	0,013876	0,011186	0,012433	0,011581
MSE TEST	0,022657	0,101409	0,08888	0,187316
CCR TRAIN	79,67	78,5	78,33	80,67
CCR TEST	78,05	65,85	65,85	68,78

Ilustración 1 Datos del dataset SIN

Los nodos por porcentajes de nodos son:

- **5%:** 6
- **15%:** 12
- **25%:** 30
- **50%:** 60

Los mejores valores de la experimentación han sido con un porcentaje de nodos del 5%.

Se puede observar que a mayor numero de nodos la regresión es peor, puede ser por conllevar un sobre entrenamiento de la red, al respecto como la generación del kmeans++ apenas hay cambios significativos que hagan plantearse el usarlo. Como conclusión creo que al ser un dataset con pocos patrones y sencilla a mayor numero de nodos más sobre entramiento estamos generando y una peor clasificación.

3.3.2 Dataset PARKINSONS

REGRESION KMEANS INIT RAMDOM

PORCENTAJE DE NODOS	5%	15%	25%	50%
MSE TRAIN	0,001656	0,000754	0,000408	0,000134
MSE TEST	0,002018	0,001288	0,001045	0,001155
CCR TRAIN	96,37	97,5	98,26	98,99
CCR TEST	96,54	97,06	97,36	97,51

REGRESION KMEANS INIT k-means++

PORCENTAJE DE NODOS	5%	15%	25%	50%
MSE TRAIN	0,001618	0,000777	0,000454	0,000144
MSE TEST	0,001944	0,001278	0,001187	0,001807
CCR TRAIN	96,45	97,74	98,21	99,06
CCR TEST	96,85	97,14	97,33	96,86

Ilustración 2 Datos del dataset PARKINSONS

Los nodos por porcentajes de nodos son:

- **5%:** 220
- **15%:** 660
- **25%:** 1101
- **50%:** 2203

Los mejores valores de la experimentación han sido con un porcentaje de nodos del 50%.

Se puede observar que a mayor número de nodos la regresión es mejor, puede ser que al ser más complejo el dataset a mayor número más precisa es la regresión, al respecto del kmeans++ apenas hay cambios significativos que hagan plantearse el usarlo.

Como conclusión creo que al ser un dataset con bastantes patrones y complejo a mayor número de nodos mejor ajuste.

3.3.3 Dataset QUAKE

REGRESION KMEANS INIT RAMDOM

PORCENTAJE DE NODOS	5%	15%	25%	50%
MSE TRAIN	0,028349	0,026902	0,02632	0,026151
MSE TEST	0,028323	0,032768	0,034947	0,035325
CCR TRAIN	94,54	94,73	94,73	94,73
CCR TEST	94,87	94,39	93,8	93,91

REGRESION KMEANS INIT k-means++

PORCENTAJE DE NODOS	5%	15%	25%	50%
MSE TRAIN	0,028351	0,026786	0,026295	0,02612
MSE TEST	0,028286	0,033604	0,036324	0,0308
CCR TRAIN	94,57	94,72	94,73	94,73
CCR TEST	94,83	94,25	94,28	94,13

Ilustración 3 Datos del dataset QUAKE

Los nodos por porcentajes de nodos son:

- **5%:** 81
- **15%:** 244
- **25%:** 408
- **50%:** 816

En este dataset no hay apenas mucha diferencia para considerar encarecidamente una u otra configuración, si es por cuestión computacional y complejidad seria conveniente usar la de un porcentaje de nodos mas bajo ya que es mas rápido. EN cuanto al uso de Kmeans++ volvemos a observar que no hay cambios notables que nos hagan plantearnos usar uno u otro.

3.3.4 Dataset VOTE

CLASIFICACION L1 n=10^-5

PORCENTAJE DE NODOS	5%	15%	25%	50%
MSE TRAIN	0,03681	0,006135	0,00614	0,006135
MSE TEST	0,056881	0,080734	0,07156	0,062385
CCR TRAIN	96,32	99,39	99,39	99,39
CCR TEST	94,31	91,93	92,84	93,76

MEJOR ARQUITECTURA: 50% L1

N PENALTY	1 L1	0,1 L1	0,01 L1	0,001 L1	0,0001 L1	0,00001 L1
MSE TRAIN	0,047239	0,009202	0,00614	0,006135	0,006135	0,006135
MSE TEST	0,042202	0,055046	0,06422	0,066055	0,062385	0,00367
CCR TRAIN	95,28	99,08	99,39	99,39	99,39	99,39
CCR TEST	95,78	94,5	93,58	93,39	93,76	93,76

MEJOR ARQUITECTURA: 50% L2

N PENALTY	1 L2	0,1 L2	0,01 L2	0,001 L2	0,0001 L2	0,00001 L2
MSE TRAIN	0,03681	0,021472	0,00614	0,006135	0,006135	0,006135
MSE TEST	0,051376	0,036697	0,05322	0,06422	0,06422	0,06422
CCR TRAIN	96,32	97,85	99,39	99,39	99,39	99,39
CCR TEST	94,86	96,33	94,68	93,58	93,58	93,58

Ilustración 4 Datos del dataset VOTE

Este es el primer dataset que usaremos para probar una clasificación mediante rbf, se sigue usando el porcentaje de nodos que hemos usado hasta ahora, pero hay que hacer diferentes pruebas variantes en el factor eta y la opción de penalty de LogisticRegression, tanto L1 como L2.

Los nodos por porcentajes de nodos son:

- **5%:** 16
- **15%:** 48
- **25%:** 81
- **50%:** 162

La mejor arquitectura resultante es la del 50%, donde en las diferentes pruebas siguientes variando los factores comentados apenas hay diferencia significativa al usar uno u otro parámetro.

Se puede observar que cuando el valor de eta es menor e igual a 0.01 esto hace que se llegue a un sobre entrenamiento del modelo dando un CCR en train cercano al 100%, también a coste computacional es una carga mas pesada tardando mas del doble en calcular los valores mas chicos de eta con los mas grandes que hemos probado. En cuanto al uso del valor L1 u L2 no hay cambios significativos.

```
#TODO: Completar el código de la función
#Penalty sed to specify the norm used in the penalization

logreg1 = LogisticRegression(penalty='l2', C=1 / eta, fit_intercept=False,multi_class='auto',solver='liblinear',max_iter=500)

logreg2 = LogisticRegression(penalty='l1', C=1 / eta, fit_intercept=False,multi_class='auto',solver='liblinear',max_iter=500)



#Entrenated model
if train_outputs.shape[1]==1:
        train_outputs = np.ravel(train_outputs)
logreg1.fit(matriz_r, train_outputs)
logreg2.fit(matriz_r, train_outputs)
print("UNO")
print(logreg1.coef_.size)
print("UNO")
print(logreg2.coef_.size)
print("TRES")
print(logreg1.coef_-logreg2.coef_)
raw_input()
return logreg1
```

Ilustración 5 Calculo de coeficientes usando L1 vs L2

En la **Ilustración 5** se observa como se han calculado el numero de coeficientes en código del dataset VOTE.

En el caso de VOTE la cantidad de coeficientes de la Regresión Logística han sido 164 usando un **r** de 10%, a continuación, también se muestra la diferencia de esos mismos coeficientes al usar penalty L1 vs L2.

```
[[ -0.34979224  -1.21567147  -2.08851421   0.88698044   1.19250259
    0.14518568   4.00766393  -1.24107146  -2.36115441   0.47173337
    0.23718865   1.16529193  -4.54821099  -0.65998007   1.24534981
   -1.41605032   1.14748425  -1.68358866   0.05185522   1.95789564
    0.26477156   2.97639973  -1.31976164  -2.07381857  -1.56426147
    0.42583996  -1.40411353  -1.16274599   1.44121749  -0.56068882
    1.29467084  -1.30761208  -2.17560333   6.83186129  -5.94441847
   -1.44948047  -1.05573656   2.23080787  -2.38707885   2.14934106
    0.62463062   1.96904268   4.14171079   1.55805352   0.86456266
   -2.7368468   -0.91668277  -0.78927202   1.20275603   0.60184369
    3.25768822   1.67285073  -1.04761843   0.15094553  -0.48703518
   -2.42271964  -0.53551951   0.62047126  -5.71784906   1.01965535
   10.20397476  -0.51414151  -1.44545168  -4.25847118  -4.65341126
    1.45604703  -4.9259092   -2.37217086   0.64465595  -0.75274372
    0.03191225   4.04883206   0.216073     1.50143105  -0.96508025
   -6.01897549   0.12006633 -10.57896165   0.66552748   3.44351665
    1.66905075  -0.81044156   3.38175229   0.77622092  -6.95106889
    6.36191151  -3.07470253   0.63851151   0.27161384  -9.82814748
    2.81620484  -0.76949509  -2.8998691    0.94902383   5.55300465
   -2.10755052   1.64330436   5.96682981  -1.00242925   2.75901492
    1.01760324  -2.70485282  -0.16595853 -17.01405305  -0.29861366
    0.12203946   0.71182317  -2.28849999   1.28029131  -0.93265828
   -1.8266528    6.21156677  -4.44595538   1.70209521   3.53293393
    0.09614615   0.81152555   0.43797669   5.39182598   1.08834348
    0.06794664   0.41524467   0.41118018  -0.88239789  -0.91141325
    0.4498374   -1.4962324   -1.474563     0.87274799   1.43070457
   -1.8991563   -0.67248177   1.48031854  -0.75997845 -17.18939693
   -1.0969235   -0.1095211   -1.43451009   3.03222228   1.83835187
    0.18767994  -7.00864477  -3.11415014   1.70130649   0.75563321
    3.93742885   2.88033587   3.00609973  -2.74662158  -0.44903722
   -2.7572309    0.75317572  -0.07156337  -0.47448426   0.73829087
   -0.07773277  -0.83311913  -0.64992327   1.1105202  -12.7087795
   -0.68118404  -3.54628893   1.49808193   6.65010078]]
```

Ilustración 6 Diferencias entre L1 y L2 en los coeficientes del dataset VOTE

3.3.5 Dataset NOMNIST

CLASIFICACION L1 n=10^-5

PORCENTAJE DE NODOS	5%	15%	25%	50%
MSE TRAIN	0,036211	0,000256	3,3E-05	0,000018
MSE TEST	0,033067	0,04327	0,03987	0,035127
CCR TRAIN	88,11	100	100	100
CCR TEST	88,67	86,2	86,73	88

MEJOR ARQUITECTURA: 50% L1

N PENALTY	1 L1	0,1 L1	0,01 L1	0,001 L1	0,0001 L1	0,00001 L1
MSE TRAIN	0,042464	0,011048	0,00089	0,000062	0,000021	0,000018
MSE TEST	0,0398456	0,023821	0,03004	0,0338045	0,35009	0,035127
CCR TRAIN	85,98	96,96	99,98	100	100	100
CCR TEST	89,47	91,33	88,93	88,13	87,93	88

MEJOR ARQUITECTURA: 50% L2

N PENALTY	1 L2	0,1 L2	0,01 L2	0,001 L2	0,0001 L2	0,00001 L2
MSE TRAIN	0,03539	0,01818055	0,00636	0,000841	0,000043	0,000002
MSE TEST	0,034311	0,024515	0,02489	0,030487	0,034533	0,036739
CCR TRAIN	88,69	94,67	98,79	100	100	100
CCR TEST	90,87	91,6	90,33	88,33	88	87,6

Ilustración 7 Datos del dataset NOMNIST

Este es el segundo dataset que usaremos para probar una clasificación mediante rbf, se sigue usando el porcentaje de nodos que hemos usado hasta ahora, pero hay que hacer diferentes pruebas variantes en el factor eta y la opción de penalty de LogisticRegression, tanto L1 como L2.

Los nodos por porcentajes de nodos son:

- **5%:** 45
- **15%:** 135
- **25%:** 225
- **50%:** 450

Se puede observar que cuando el valor de eta es menor e igual a 0.0001 esto hace que se llegue a un sobre entrenamiento del modelo dando un CCR en train cercano al 100%, también a coste computacional es una carga más pesada tardando más del doble en calcular los valores más chicos de eta con los más grandes que hemos probado. En cuanto al uso del valor L1 u L2 no hay cambios significativos.

La mejor arquitectura resultante es la del 50%, donde en las diferentes pruebas siguientes variando los factores comentados apenas hay diferencia significativa al usar uno u otro parámetro, la diferencia se encuentra en el tiempo que tarda en ejecutar una u otra configuración.

```
[[47  0  0  1  1  1]
 [ 2 40  0  3  4  1]
 [ 0  1 45  0  3  1]
 [ 2  6  0 41  0  1]
 [ 0  0  2  0 43  5]
 [ 2  0  1  1  2 44]]
```

Ilustración 8 Matriz de confusión de la mejor configuración de NOMNIST en RBF

Algunas de las letras que no clasifico bien:

Como se puede observar son difíciles de reconocer para el clasificador pues algunas tienen algunas peculiaridades.

De la practica 2 podemos encontrar:

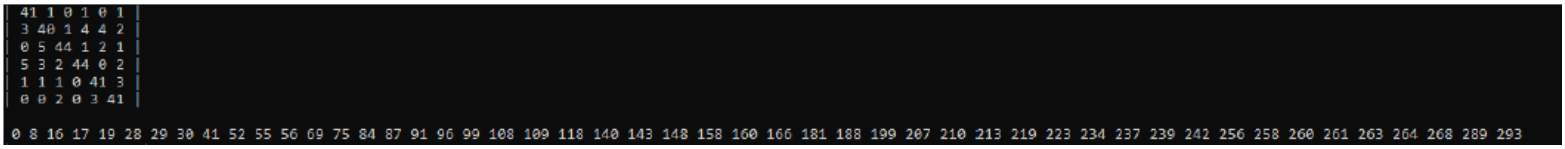

Ilustración 9 Matriz de confusión practica 2 y fallos.

Podemos observar que clasifican semejante al comparar las matrices de confusión.

Tiempos medios por ejecuciones sin valor de eta:

- **5%:** 1.83 segundos
- **15%:** 2.22 segundos
- **25%:** 2.68 segundos
- **50%:** 3.25 segundos

Tiempos medios por ejecuciones modificando factor eta

Nodos/ETA	1	0.1	0.01	0.001	0.0001	0.00001
5%	1.82 s	2.16 s	2.20	2.39 s	2.37 s	2.31 s
15%	1.97 s	2.46 s	6.02 s	15.18 s	18.71 s	19.29 s
25%	2.27 s	2.86 s	6.46 s	8.75 s	9.32 s	9.14 s
50%	2.50 s	3.69 s	6.22 s	6.31 s	6.46 s	6.53 s

En la **Ilustración 5** se observa cómo se han calculado el número de coeficientes en código del dataset VOTE.

En el caso de NOMNIST la cantidad de coeficientes de la Regresión Logística han sido 546 usando un **r** de 10%.

6. Lanzada de dataset VOTE como regresión

Se ha lanzado el dataset de Vote con los siguientes parámetros:

```
python.exe ./rbf.py -t ./csv/train_vote.csv -T ./csv/test_vote.csv -r 0.15 -e 0.00001
   **********
   Resumen de resultados
   **********
   Tiempo medio: 0.494001
   MSE de entrenamiento: 0.029579 +- 0.000757
   MSE de test: 0.042657 +- 0.002684
   CCR de entrenamiento: 96.87% +- 0.23%
   CCR de test: 94.13% +- 1.10%
```

Ilustración 10 Lanzada de dataset VOTE como un problema de regresión

Si comparamos la **Ilustración 8** con la **Ilustración 4** con los mismos parámetros que en esta lanzada podemos ver que son semejantes en ambos casos.

Se ha intentado también con el dataset de nomnist, pero los datos difieren bastante al usarlo como regresor.

7. Conclusiones

Las redes de base radial son un tipo de redes de neuronas artificiales que calculan la salida de la función en función de la distancia a un punto denominado centro. Al igual que con los perceptrones multicapa, sirven como aproximadores universales.

La función de base radial es una función que calcula la distancia euclídea de un vector de entrada x respecto de un centro c. A cada neurona de la capa de entrada le corresponde una función de base radial y un peso de salida wi. El patrón de salida ingresa a una neurona de salida que suma las entradas y da como resultado una salida.

Las redes RBF tienen una construcción rígida de tres capas: Capa de entrada, capa oculta y capa de salida (a diferencia de otras redes backpropagation).

Estas redes se pueden implementar en el día a día en ámbitos como análisis de imágenes, reconocimiento de la voz, diagnósticos médicos, regresión.

Aunque rbf es muy polivalente y una de las mas usadas en la actualidad no es necesariamente la mas potente ni la mejor ya que tiene algunas limitaciones como suele ser el proceso de aprendizaje para ciertos problemas complejos.

En la practica hemos visto que el ajuste a los problemas dado ha sido mejor de lo esperado y que incluso en algunos de usar una red simple hemos usado algún parámetro que ha logrado hacer converger más rápido el algoritmo o mejorar la puntuación final de la función objetivo, que es la del mejor ajuste para todos los patrones de entrada de cada uno de los dataset usados.

Es realmente relevante saber realizar estudios de diferentes valores de nodos y parámetros para los diferentes problemas ya que esto es determinante para el resultado final del problema y sobre todo para el coste computacional que es necesario para ciertos parámetros.

8. Ejercicio Jupyter Notebook

```python
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import MinMaxScaler
def principal():
    digits=load_digits()

    #DIvidimos en conjunto de test y train
    train_inputs,test_inputs,train_outputs,test_outputs=train_test_split(digits['data'],digits['target'])
    scaler=MinMaxScaler()
    scaler.fit(train_inputs)
    scaler.fit(test_inputs)

    #Algoritmo KNN para vecinos
    print('------------Ejecutando KNN para i vecinos---------')
    for i in range(1,5):
        print('Vecinos: ',i)
        knn = KNeighborsClassifier(n_neighbors=i)
        knn.fit(train_inputs,train_outputs)
        print('CCR Train')
        print(knn.score(train_inputs,train_outputs))
        print('CCR Test')
        print(knn.score(test_inputs,test_outputs))
        print('|d')

    #Para problemas multiclase utilizamos newton-cg y multiclass ovr para que se entrene un poblema binario para cada clase
    logreg = LogisticRegression(random_state=0, solver='newton-cg',multi_class='ovr', max_iter=500)
    logreg.fit(train_inputs, train_outputs)
    print('CCR Train')
    print(logreg.score(train_inputs, train_outputs))
    print('CCR Test')
    print(logreg.score(test_inputs, test_outputs))
    print('-------------------')

if __name__ == "__main__":
    principal()
```

Ilustración 11 Ejercicio Jupyter Notebook

9. Índice de ilustraciones