Centro de investigación en Computación

Clasificación inteligente de Patrones

Métricas de Minkowski, complejidad de datos, conversión e imputación

Autor:
Diego Noguez Ruiz

Profesor:
Dr. Cornelio Yáñez Márquez

15 de mayo de 2023

Índice general

1. Introducción

Estamos llegando al final del curso CIP y hemos obtenido una gran cantidad de conocimientos acerca de los cuatro elementos básicos de un cip.

Hasta el momento, hemos explorado dos clasificadores: el clasificador euclidiano y el k-NN. Es importante tener en cuenta que solo hemos utilizado el k-NN con la distancia euclidiana, por lo que es necesario ampliar el algoritmo con distintas métricas.

En la vida real, nos enfrentamos a diversos tipos de datasets, como aquellos que tienen una alta complejidad de datos. Por lo tanto, en nuestra última sesión, hemos estudiado cómo abordar estos casos para mejorar la calidad de nuestros modelos.

En este documento, pondremos en práctica las métricas de Manhattan, Euclidiana y Chebyshov, además de explorar técnicas para datasets con datos mezclados y valores perdidos. Esto nos permitirá ampliar nuestro conocimiento y habilidades en la creación y análisis de modelos de aprendizaje automático.

2. Desarrollo

2.1. Propósito de la tarea

- Ejemplificar el uso de las tres métricas de Minkowski más importantes y comparar los resultados al clasificar patrones.
- Ejemplificar algunas métricas diferentes a las de Minkowski.
- Ejecutar conversión e imputación en datasets a fin de aplicar clasificadores basados en métricas.

2.2. Parte 1

Instrucciones: Usando el dataset sonar, realizar lo siguiente: con el método de validación Leave-one-out, calcular las tres matrices de confusión que resultan de aplicar el algoritmo 1-NN implementado con cada una de las tres métricas de Minkowski más importantes: city block, euclidiana y chessboard. Para los tres casos, y SIN USAR LA BIBLIOTECA scikit-learn, reprotar el código en python y las medidas de desempeño estudiadas en el curso CIP.

Primero definimos la clase positiva como la 1.

A continuación las respectivas métricas y medidas de desempeño. Es importante notar que el dataset es balanceado $IR = 1.14$. Por otro lado usaremos la siguiente notación para la matriz de confusión: $CM = [[TP, FN], [FP, TN]]$.

- Manhattan:
  CM=$[[98, 13], [18, 79]]$.
  - Accuracy:0.851
  - Recall:0.8829
  - Specificity: 0.8144
  - Balanced Accuracy:0.8487
  - Precision: 0.8448
  - f1: 0.8634
  - MCC: 0.0
- Euclidiana:
  CM=$[[96, 15], [21, 76]]$
  - Accuracy:0.8269
  - Recall:0.8649
  - Specificity: 0.7835
  - Balanced Accuracy:0.8242
  - Precision: 0.8205
  - f1: 0.8421
  - MCC: 0.0
- Chebyshov:
  CM=$[[96, 15], [28, 69]]$
  - Accuracy:0.7933
  - Recall:0.8649
  - Specificity: 0.7113
  - Balanced Accuracy:0.7881
  - Precision: 0.7742
  - f1: 0.817
  - MCC: 0.0

Coding

El código fue realizado en Python.

Primero se definieron funciones de las respectivas métricas.
Minkowski:

```python
# a and b should be vectors of the same lenght to compute the distance

# k is the value I want for the metric of Minkowski

def mink_metric(k,a,b):

    n=len(a)

    s=0 #sum

    for i in range(n):
        value= (abs(a[i]-b[i]))**k

        s=s+value

    s= s**(1/k)

    return s
```

Chebyshov:

```python
def chebyshev_metric(a,b):

    n=len(a)

    values=[]

    for i in range(n):
        value= (abs(a[i]-b[i]))

        values.append(value)

    s= max(values)

    return s
```

Definimos una función general para aplicar k-nn con LOOCV con una métrica en particular (Minkowski o Chebyshov) con un caso particular de patrón de prueba. Usando esta función se realizará un loop para usar todos los patrones.

```python
from collections import Counter
import math

#K-NN with Leave One Out Cross Validation
def knn_LOOCV(k,minkowski_value,df_validation,class_positive,class_negative,index_test,last_positive_index,TP,FN,FP,TN):

```

```
    train_df = df_validation #dataset where the first
        patterns are all positive class and the second part
        are all negative class

    pattern_index = index_test
    test_df = df_validation[pattern_index]

    n=last_positive_index

    if pattern_index<=n:
        class_of_test_pattern = class_positive
    else:
        class_of_test_pattern = class_negative

    distances = []

    for i in range(len(train_df)):

        #the lines below correspond to different distances
        #just use the metric i need to use

        #dist = round(mink_metric(minkowski_value,test_df,
            train_df[i]),4) #minkowski metric

        dist = round(chebyshev_metric(test_df,train_df[i])
            ,4) #Chebyshev metric

        if i<=n:
            distances.append([dist,i+1,class_positive]) #it
                need to be sum one because of the python
                numeration
        else:
            distances.append([dist,i+1,class_negative])

    del distances[pattern_index] #delete the list of
        distance from the test pattern to itself

    kNN=k
    sorted_dist = sorted(distances, key=lambda x: x[0])[:kNN
        ]

    #now i want to count the most repeated class from the
        KNN list

    # extract the third element from each sublist and put
        them in a separate list
    third_elements = [sublist[2] for sublist in sorted_dist]

    # use Counter to count the frequency of each value in
        the third_elements list
```

```
    counter = Counter(third_elements)

    most_common_class = counter.most_common(1)[0][0] #this
        print the most common class

    if most_common_class == class_of_test_pattern:


        if class_of_test_pattern==class_positive:
            TP +=1
        elif class_of_test_pattern==class_negative:
            TN +=1
    else:
        #print("\nFue un \\textbf{ERROR}")

        if class_of_test_pattern==class_positive:
            FN +=1
        elif class_of_test_pattern==class_negative:
            FP +=1
    return TP,FN,FP,TN
```

A continuación el loop generalizado. Este código fue la plantilla usada para lás 3 métricas, solo haciendo los respectivos cambios en las entradas de *minkowski_value*, o bien cambiando la variable *dist* en el código con la métrica Chebyshov.

```
class_positive=1
class_negative=2

n=last_pos_index

TP=0
FN=0
FP=0
TN=0

#minkowski with m=1
for i in range(len(df)):
    TP,FN,FP,TN=knn_LOOCV(1,1,df,class_positive,
        class_negative,i,n,TP,FN,FP,TN)


CM=[[TP,FN],[FP,TN]]
print(CM)
```

2.3. Parte 2

Instrucciones: Realizar las siguientes acciones, en estricto orden, sobre el dataset Congressional Voting Record (https://sci2s.ugr.es/keel/dataset.php?cod=87)

- Usando la moda por clase, imputar los valores perdidos.

  Sea df la variable donde esta guardado el dataset como pandas.

```
#shape give to me a tuple (rows,columns)
num_col= df.shape[1]
num_rows= df.shape[0]

#Imputation
import statistics

for col in range(num_col-1):

    mode = statistics.mode(df[col])

    for row in range(num_rows):
        if df[col][row]=='?':
            df[col][row]=mode
```

- Aplicar Label Coding, convirtiendo la etiqueta “y” en 1 y la etiqueta “n” en 2.

```
#label Coding
for col in range(num_col-1):

    for row in range(num_rows):
        if df[col][row]=='y':
            df[col][row]=1
        else:
            df[col][row]=2
```

- Con el método de validación Leave-one-out, calcular la matriz de confusión que resulta de aplicar el algoritmo 1-NN al dataset Congressional Voting Record con la métrica euclidiana. SIN USAR LA BIBLIOTECA scikit-learn, repotar el código en python y balanced accuracy.

  Usaremos la misma función usada anteriormente de *knn_LOOCV* con la métrica euclidiana. Adicionalmente supongamos que la clase positiva es la *republican*

```
class_positive='republican'
class_negative="democrat"

n=last_pos_index

TP=0
FN=0
FP=0
TN=0
```

```
10
11 for i in range(len(data)):
12     TP,FN,FP,TN=knn_LOOCV(1,data,class_positive,
           class_negative,i,n,TP,FN,FP,TN)
13
14
15 CM=[[TP,FN],[FP,TN]]
16 print(CM)
```

Obteniendo la siguiente matriz de confusión:

$$[[161, 7], [35, 232]]$$

Por otro lado sabemos que $IR = 1.58$, por lo tanto el dataset es desbalanceado. Procediendo a calcular la medida de desempeño solicitada.

```
1 def medidas(CM):
2     recall = round((CM[0][0]) / (sum(CM[0])),4)
3     specificity = round((CM[1][1])/sum(CM[1]),4)
4     balanced_acc= round((recall+specificity)/2,4)
```

- Recall:0.9583
- Specificity: 0.8689
- **Balanced Accuracy**:0.9136

2.4. Parte 3

- Describir las cuatro propiedades que debe cumplir una métrica.

  Definición 1. *Sea (X, d) un espacio métrico. Una función $d : X \times X \to \mathbb{R}$ se llama una métrica si y solo si satisface:*

  1. ***Definida positiva:*** *$d(x, y) \geq 0$ para todo $x, y \in X$.*
  2. *$d(x, y) = 0$ si y sólo si $x = y$.*
  3. ***Simetría:*** *$d(x, y) = d(y, x)$ para todo $x, y \in X$.*
  4. ***Desigualdad triangular:*** *$d(x, z) \leq d(x, y)+d(y, z)$ para todo $x, y, z \in X$.*

- Con las propiedades del inciso previo, demostrar que la métrica discreta efectivamente sí es métrica.

  Teorema 2. *Sea X un conjunto y $x, y \in X$, entonces*

  $$d(x, y) = \begin{cases} 0 & si\ x = y, \\ 1 & si\ x \neq y. \end{cases}$$

  Es una métrica.

Demostración. Para demostrar que es métrica se deben satisfacer las 4 propiedades de la definición de métrica. Procederemos a demostrar que se satisfacen las 4 propiedades.

1. Trivial por definición de $d(x, y)$.
2. Trivial por definición de $d(x, y)$.
3. Si $x = y$, entonces $d(x, y) = d(y, x) = 0$. Si $x \neq y$, entonces $d(x, y) = 1$ y $d(y, x) = 1$, por lo tanto $d(x, y) = d(y, x)$.
4. - Caso 1: Si $x = y$ o $y = z$, entonces trivialmente se cumple que $d(x, z) \leq d(x, y) + d(y, z)$
   - Caso 2: Si $x \neq y \neq z$, entonces $d(x, y) = d(y, z) = 1$ y $d(x, z)$ es 0 (si $x = z$) o bien es 1 en caso contrario, en cualquiera de los casos $d(x, z) \leq d(x, y) + d(y, z) = 2$

□

3. Conclusiones y trabajo futuro

En conclusión, este trabajo ha demostrado la importancia de conocer diversas métricas y como estas pueden influir en la elección y evaluación de diferentes algoritmos. En particular, la definición de las métricas de Minkowski ha ampliado significativamente el rango de clasificadores que podemos aplicar, enriqueciendo nuestro conocimiento en este campo. Además, realizar el código de la generalización del k-NN en Python en una tarea anterior ha simplificado nuestro trabajo en esta ocasión y nos ha permitido enfocarnos en la evaluación de diferentes métricas.

Por otro lado, me ha gustado saber como proceder cuando tenemos dataset con una nueva complejidad en sus datos.

Espero seguir explorando nuevos clasificadores en las próximas semanas para ampliar aún más nuestro conocimiento y estar al tanto de las últimas tendencias en el estado del arte. Espero también poder profundizar en el estudio de los clasificadores más utilizados en la actualidad y seguir mejorando en la implementación de algoritmos en Python.

Referencias