Aprendizaje Automático
Tarea

Carlos Alexis Barrios Bello
zS23000636@estudiantes.uv.mx

Maestría en Inteligencia Artificial

IIIA Instituto de Investigaciones en Inteligencia Artificial
Universidad Veracruzana
Campus Sur, Calle Paseo Lote II, Sección 2a, No 112
Nuevo Xalapa, Xalapa, Ver., México 91097

26 de abril de 2024

1. Introducción

La tarea de extraer toda la información de bases de datos se le encarga a los algoritmos empleados en aprendizaje automático (o en su gran mayoría). La mayoría de estos algoritmos solo pueden aplicarse a datos descritos por atributos numéricos discretos o nominales. En el caso de atributos continuos, se necesita hacer la discretización con ayuda de algún algoritmo que transforme los atributos continuos en atributos discretos.

La discretización transforma los valores de un atributo continuo en un número finito de intervalos y asocia con cada intervalo un valor numérico discreto. Para datos de tipo mixto (continuos y discretos), la discretización generalmente se realiza antes del proceso de aprendizaje. La discretización se puede dividir en dos tareas. La primera tarea es encontrar el número de intervalos discretos. Solo algunos algoritmos de discretización realizan esto; a menudo, el usuario debe especificar el número de intervalos o proporcionar una regla heurística. La segunda tarea es encontrar el ancho o los límites para los intervalos, dados el rango de valores de un atributo continuo. Kurgan and Cios (2004)

2. Marco Teórico

2.1. Algoritmo de CAIM

El algoritmo CAIM (maximización de la interdependencia entre clase y atributo), el cual está diseñado para trabajar con datos supervisados. El objetivo éste es maximizar la interdependencia entre la clase y el atributo y generar un número mínimo de intervalos discretos. A diferencia de algunos otros algoritmos de discretización, este algoritmo no requiere que el usuario predefina el número de intervalos Kurgan and Cios (2004).
El esquema de discretización óptimo se puede encontrar buscando en el espacio de todos los esquemas de discretización posibles para encontrar aquel con el valor más alto del criterio CAIM. Tal búsqueda de un esquema con el valor óptimo global de CAIM es altamente combinatoria y consume mucho tiempo. Por lo tanto, el algoritmo CAIM utiliza un enfoque codicioso, que busca el valor aproximadamente óptimo del criterio CAIM encontrando los valores máximos locales del criterio. Aunque este enfoque no garantiza encontrar el máximo global, es computacionalmente económico y se aproxima bien a encontrar el esquema de discretización óptimo, como se muestra en la sección de resultados. El algoritmo consta de estos dos pasos:

1. Inicialización de los límites de intervalo candidatos y el esquema de discretización inicial.

2. Adiciones consecutivas de un nuevo límite que resulta en el valor localmente más alto del criterio CAIM.

A continuación el pseudocódigo del algoritmo de CAIM Kurgan and Cios (2004):
Given: Data consisting of M examples, S classes, and continuous attributes F_i
For every F_i do:

1.
   - 1.1 find maximum (d_n) and minimum (d_0) values of F_i
   - 1.2 form a set of all distinc values of F_i in ascending order, and initialize all possible interval boundaries B with minimum, maximum and the midpoints of all the adjacent pairs in the set
   - 1.3 set the initial discretization scheme as $D : \{[d_0, d_n]\}$, set GlobalCAIM=0

2.
   - 2.1 initialize $k = 1$

- 2.2 tentatively add an inner boundary, which is not already in D, from B, and calculate correspoding CAIM value
- 2.3 after all the tentative additions have been tried accepted the one with highest value of CAIM
- 2.4 if ($CAIM > GlobalCAIM$ *or* $k < S$) then update D with the accepted in step 2.3 boundary and set $GlobalCAIM = CAIM$, else terminate
- 2.5 set $k = k + 1$ and go to 2.2

Output: Discretization scheme D
El algoritmo comienza con un solo intervalo que cubre todos los posibles valores de un atributo continuo, y lo divide de forma iterativa. De todos los puntos de división posibles que se prueban (con reemplazo) en el paso 2.2, elige el límite de división que proporciona el valor más alto del criterio CAIM. El algoritmo asume que cada atributo discretizado necesita al menos el número de intervalos igual al número de clases, ya que esto asegura que el atributo discretizado pueda mejorar la clasificación posterior. Kurgan and Cios (2004)
El algoritmo CAIM implementa un equilibrio entre un costo computacional razonable y encontrar el esquema de discretización óptimo. A pesar de la manera codiciosa en la que trabaja el algoritmo, los esquemas de discretización que genera tienen una interdependencia entre clase y atributo muy alta y siempre un pequeño número de intervalos de discretización. Para los conjuntos de datos utilizados en la sección experimental, el algoritmo CAIM genera esquemas de discretización con el (posiblemente) menor número de intervalos que aseguran un bajo costo computacional, y siempre logra una interdependencia entre clase y atributo muy alta, lo que resulta en una mejora significativa en la tarea de clasificación posteriormente realizada. Kurgan and Cios (2004)

3. Bases de datos utilizadas

- **Iris dataset:** El conjunto de datos contiene 3 clases de 50 instancias cada una, donde cada clase se refiere a un tipo de planta de iris. Una clase es linealmente separable de las otras 2; estos últimos no son linealmente separables entre sí.Fisher (1988)

- **Dry bean dataset:** Se utilizaron siete tipos diferentes de frijoles secos, teniendo en cuenta características como forma, tipo y estructura

según la situación del mercado. Se desarrolló un sistema de visión por computadora para distinguir siete variedades diferentes registradas de frijol seco con características similares con el fin de obtener una clasificación uniforme de las semillas. Para el modelo de clasificación se tomaron imágenes de 13,611 granos de 7 diferentes frijoles secos registrados con una cámara de alta resolución. Las imágenes de frijoles obtenidas mediante un sistema de visión por computadora se sometieron a etapas de segmentación y extracción de características, y se obtuvieron un total de 16 características; A partir de los granos se obtuvieron 12 dimensiones y 4 formas. bea (2020)

- **Glass identification dataset:** El estudio de la clasificación de los tipos de vidrio estuvo motivado por la investigación criminológica. En la escena del crimen, el cristal que queda puede utilizarse como prueba... ¡si se identifica correctamente! German (1987).

- **Letter Recognition dataset:** El objetivo es identificar cada una de una gran cantidad de pantallas de píxeles rectangulares en blanco y negro como una de las 26 letras mayúsculas del alfabeto inglés. Las imágenes de los personajes se basaron en 20 fuentes diferentes y cada letra dentro de estas 20 fuentes se distorsionó aleatoriamente para producir un archivo de 20.000 estímulos únicos. Cada estímulo se convirtió en 16 atributos numéricos primitivos (momentos estadísticos y recuentos de bordes) que luego se escalaron para ajustarse a un rango de valores enteros del 0 al 15. Slate (1991)

- **Seeds dataset:** El grupo examinado estaba formado por granos pertenecientes a tres variedades diferentes de trigo: Kama, Rosa y Canadiense, 70 elementos cada uno, seleccionados al azar para el experimento. Se detectó una visualización de alta calidad de la estructura interna del núcleo mediante una técnica de rayos X suaves. Las imágenes se registraron en placas de rayos X KODAK de 13x18 cm. Charytanowicz Magorzata and Szymon (2012)

- **MAGIC Gamma Telescope dataset:** Los datos son generados por Monte Carlo para simular el registro de partículas gamma de alta energía en un telescopio Cherenkov atmosférico. Bock (2007)

- **Rice (Cammeo and Osmancik) dataset:** Se tomaron un total de 3810 imágenes de granos de arroz para las dos especies, se procesaron e inferencias de características se realizaron. Se obtuvieron 7 características morfológicas para cada grano de arroz. ric (2019)

- **Wine Quality:** Los dos conjuntos de datos están relacionados con las variantes roja y blanca del vino "Vinho Verde" portugués. Cortez Paulo and J. (2009)

- **Yeast dataset:** Una base de datos simple que contiene 17 atributos de valor booleano. El atributo "type" parece ser el atributo de clase. Aquí hay un desglose de qué animales están en qué tipo. Nakai (1996)

4. Codificación

En esta sección, sólo se presentará el código de CAIM realizado con base al artículo Kurgan and Cios (2004), para las implementaciones de Naive Bayes, ID3, k-folds CV y k-folds CV estratificado véase en los códigos del .zip mandado.

```
import pandas as pd
import numpy as np

#Se efine una función para calcular el valor CAIM para
#una quanta matriz
def calculate_caim(quanta_matrix):
    max_values = np.max(quanta_matrix, axis=0)
    sums = np.sum(quanta_matrix, axis=0)
    #Se calcula el valor CAIM y evita la división por
    #cero verificando si la suma  total no es cero. Si es
    #cero, el valor CAIM es 0.
    caim = np.sum((max_values ** 2) / np.where(sums == 0, 1, sums)) /
    len(sums) if np.sum(sums) != 0 else 0
    return caim

#Se define una función para crear la matriz quanta, ésta cuenta
#cuántos datos de cada clase caen en cada intervalo de un atributo.
def create_quanta_matrix(data, attribute, intervals, classes,
    class_label):
    #Se inicializa una matriz de ceros con una fila por clase
    #y una columna por cada intervalo, y se usa menos uno, ya
    #que los intervalos definen límites
    quanta_matrix = np.zeros((len(classes), len(intervals) - 1))
    for idx, cl in enumerate(classes):
        class_data = data[data[class_label] == cl][attribute]
```

```
        for i in range(1, len(intervals)):
            #Aquí se cuentan los valores de cada clase que
            #caen en cada intervalo
            quanta_matrix[idx, i - 1] = class_data[(class_data
            >= intervals[i - 1]) & (class_data <
            intervals[i])].count()
    return quanta_matrix

##Se define una función para la discretización CAIM sobre un
#conjunto de datos.
def caim_discretization(data, attribute, class_label):
    values = data[attribute].dropna().unique()
    classes = data[class_label].unique()
    #Se pone esta impresión porque hubo problemas con la lectura
    #de la clase de las bases de datos
    print(f'Classes found: {classes}')

    #Se aplica el step 1 del algoritmo de CAIM
    min_value = np.min(values)
    max_value = np.max(values)
    values_sorted = np.sort(values)
    mid_points = (values_sorted[:-1] + values_sorted[1:]) / 2
    boundaries = np.concatenate(([min_value], mid_points, [max_value]))
    intervals = [min_value, max_value]

    global_caim = 0

    #Ahora se aplica el step 2
    #Se inicia un bucle para encontrar el mejor intervalo adicional.
    while True:
        best_caim = global_caim
        best_interval = None

        for boundary in boundaries:
            if boundary not in intervals:
                test_intervals = sorted(intervals + [boundary])
                #Se genera la matriz quanta para los intervalos de prueba.
                quanta_matrix = create_quanta_matrix(data, attribute,
                test_intervals, classes, class_label)
                #Se alcula el valor CAIM para estos intervalos de prueba.
```

```
            caim_value = calculate_caim(quanta_matrix)

            if caim_value > best_caim:
                best_caim = caim_value
                best_interval = boundary
    #Se verifica si se encontró un intervalo mejor.
    if best_interval is not None and best_caim > global_caim:
        intervals.append(best_interval)
        intervals = sorted(intervals)
        global_caim = best_caim
    else:
        break

data[attribute] = pd.cut(data[attribute], bins=intervals,
labels=range(len(intervals) - 1), include_lowest=True,
right=True)

return intervals
```

5. Conceptos Pedregosa et al. (2011)

- **GaussianNB:**

1. *¿Qué es?* Es una técnica de aprendizaje automático basada en el teorema de Bayes, que asume que la probabilidad de que ocurra un evento dado es influenciada por la probabilidad de eventos anteriores. En el contexto de GaussianNB, la suposición es que las características (o variables de entrada) de los datos siguen una distribución normal (gaussiana).
2. *¿Cómo funciona?* El algoritmo GaussianNB funciona bajo el supuesto de que las características de los datos son independientes entre sí dado el resultado, y que cada característica se distribuye normalmente. Este supuesto de independencia simplifica los cálculos, lo que permite modelar la probabilidad de los resultados simplemente con el producto de las probabilidades individuales de cada característica.
3. *Uso:* El principal uso de GaussianNB es para clasificación. Dado un conjunto de datos de entrenamiento, GaussianNB calcula la

media y la desviación estándar para cada clase y característica. Luego, utiliza estos parámetros para calcular la probabilidad de que nuevos ejemplos pertenezcan a cada clase. El modelo clasifica cada nuevo ejemplo asignándolo a la clase con la mayor probabilidad logarítmica.

4. *¿Qué tipo de datos acepta?* Entradas: Acepta características numéricas que se supone siguen una distribución normal. Si los datos no son normalmente distribuidos, el rendimiento del clasificador puede degradarse.
   Salidas: Puede manejar múltiples clases, no sólo clasificaciones binarias.

- **k-folds with cross validation simple (KFold in Python):**

1. *¿Qué es?* KFold es una función de Scikit-learn utilizada para dividir un conjunto de datos en múltiples subconjuntos (o "folds") para realizar validaciones cruzadas.
2. *¿Cómo funciona?* KFold divide todo el conjunto de datos en k subconjuntos consecutivos. Luego, el modelo se entrena en $k-1$ de estos folds, mientras que el fold restante se utiliza como conjunto de prueba. Este proceso se repite k veces, con cada uno de los k folds utilizado exactamente una vez como conjunto de prueba. Al final, se tienen k resultados de rendimiento del modelo, que generalmente se promedian para obtener una medida más robusta de la eficacia del modelo.
3. *Uso:*
   - Estimar la precisión de un modelo en un conjunto de datos.
   - Reducir la variabilidad de la estimación al promediar los resultados de múltiples iteraciones de entrenamiento y prueba sobre diferentes divisiones de los datos.
   - Ayudar en la selección y ajuste de modelos, permitiendo comparar diferentes configuraciones y elegir la que mejor rendimiento ofrezca bajo validación cruzada.
4. *¿Qué tipo de datos acepta?* Puede ser utilizado con cualquier tipo de datos numéricos o categóricos, siempre y cuando estén en un formato compatible con Scikit-learn. No impone restricciones sobre la distribución de los datos o el tipo de problema.

5. *Parámetros:*
    - n_splits: Define el número de particiones que se crearán del conjunto de datos.
    - $Shuffle$: Barajea los datos antes de ser divididos.
    - $random_state$: Permite especificar un seed para generar números aleatorios, esto para obtener los mismos resultados.

- **k-folds with cross validation estratificado (StratifiedKFold in Python):**

1. *¿Qué es?* La principal diferencia entre StratifiedKFold y KFold es que StratifiedKFold mantiene la proporción de cada clase en los folds como en el conjunto de datos original. Esto es importante en situaciones donde las clases no están equilibradas, ya que asegura que cada fold sea una buena representación de todos los estratos del conjunto de datos.
2. *¿Cómo funciona?* Divide el conjunto de datos en k folds, asegurándose de que cada fold contenga aproximadamente la misma proporción de ejemplos de cada clase como el conjunto completo. Después de lo anterior hace el mismo proceso que KFold.
3. *Uso:*
    - Realizar una evaluación más precisa y estable de los modelos de clasificación, especialmente en casos de distribución desigual de clases.
    - Reducir el sesgo en la validación cruzada al mantener una representación uniforme de las clases en cada fold.
    - Mejorar la confianza en el rendimiento del modelo evaluado, ya que cada clase está adecuadamente representada en cada parte del proceso de validación y entrenamiento.
4. *¿Qué tipo de datos acepta?* Los mismos que KFold.
5. *Parámetros:* Los mismos que KFold.

6. Resultados

Antes de presentar los resultados, se resalta que para todos los resultados de precisión así como de desviación estándar, se hizo uso de $k = 10$

para todas las tablas. Se presentaron 2 tablas en total para ambos métodos (Naive Bayes e ID3), una tabla será para k-folds CV y la otra para k-folds CV estratificado, todo lo mencionado se hizo desde cero, sin usar alguna paquetería que no fuera numpy y pandas (si se tiene dudas, checar los códigos del zip).

Bases de datos discretizadas (CV sencillo):

Base de datos	ID3 (ACC)	NB (ACC)	ID3 (STD)	NB (STD)
Iris	0.9333	0.9400	0.0843	0.0554
Dry Bean	0.5951	0.8156	0.3086	0.0102
Glass	0.5573	0.6643	0.2703	0.1370
Letter	0.5599	0.4798	0.0111	0.0083
Seeds	0.9142	0.9048	0.0791	0.0369
Gamma	0.7997	0.7364	0.1194	0.0073
Rice	0.9262	0.9150	0.0244	0.0101
Wine Red	0.5121	0.5647	0.0457	0.0366
Wine White	0.4806	0.5096	0.0440	0.0270
Yeast	0.5450	0.5701	0.0521	0.0387

Bases de datos discretizadas (CV estratificado):

Base de datos	ID3 (ACC)	NB (ACC)	ID3 (STD)	NB (STD)
Iris	0.9800	0.9333	0.0305	0.0667
Dry Bean	0.8365	0.8157	0.0075	0.0095
Glass	0.8	0.6891	0.0818	0.1432
Letter	0.5662	0.4804	0.0056	0.0098
Seeds	0.9619	0.9095	0.0415	0.0333
Gamma	0.8066	0.7364	0.0081	0.0087
Rice	0.9343	0.9150	0.0108	0.0115
Wine Red	0.6356	0.5661	0.0262	0.0361
Wine White	0.5355	0.5078	0.0175	0.0179
Yeast	0.5687	0.5715	0.0299	0.0273

7. Conclusión

En esta tarea se percató que entre menos clases y varios intervalos realizados por CAIM, ID3 y NB hacen una precisión mejor, se puede ver reflejado con Iris, Seeds y Rice. Eso sí, disminuyó significativamente la desviación estándar en estos casos. Mientras que para los otros datasets se disminuyó su precisión, pero la desviación estándar aumentó un poco. También se pudo verificar que cross validation estratificado tiene una leve mejora en los resultados, tanto en ACC como en STD.

Referencias

(2019). Rice (Cammeo and Osmancik). UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5MW4Z.

(2020). Dry Bean. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C50S4B.

Bock, R. (2007). MAGIC Gamma Telescope. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C52C8B.

Charytanowicz Magorzata, Niewczas Jerzy, K. P. K. P. and Szymon, L. (2012). Seeds. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5H30K.

Cortez Paulo, Cerdeira A., A. F. M. T. and J., R. (2009). Wine Quality. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C56S3T.

Fisher, R. A. (1988). Iris. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C56C76.

German, B. (1987). Glass Identification. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5WW2P.

Kurgan, L. and Cios, K. (2004). Caim discretization algorithm. *Knowledge and Data Engineering, IEEE Transactions on*, 16:145– 153.

Nakai, K. (1996). Yeast. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5KG68.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas,

J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Slate, D. (1991). Letter Recognition. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5ZP40.