Aceleración del algoritmo K-NN

Günther Roland
Universidad Carlos III
Av. de la Universidad, 30
28911 Leganés (Madrid)
g.roland(at)student.tugraz.at

RESUMEN
En el siguiente trabajo presentaré un algoritmo K-NN iterativo que utiliza un número reducido de propiedades para acelerar la clasificación.

Términos Generales
Algorithms, Performance, Languages.

Palabras Clave
K-NN, performance, runtime

1. INTRODUCCIÓN
El K-NN es un algoritmo de aprendizaje inductivo supervisado utilizado frecuentemente debido a su sencillez y su alta capacidad de clasificar datos correctamente. Uno de los problemas del algoritmo K-NN es que necesita mucho tiempo para una clasificación a causa de su simple funcionamiento.

El reconocimiento de imágenes es una tarea bastante compleja para un procesador y requiere un alto rendimiento.

2. EL ALGORITMO K-NN
El algoritmo K-NN es un algoritmo supervisado, lo que significa que el conjunto de los datos de entrenamiento incluye, además de las propiedades multidimensionales utilizadas para el reconocimiento, clasificadores para predecir la clase de los datos de entrada.

Para clasificar, el K-NN utiliza un tipo de distancia, como, por ejemplo, la distancia euclídea, con la que determina todas las distancias entre el punto a clasificar y todos los puntos del conjunto de entrenamiento. Con las distancias calculadas determina los K vecinos más cercanos y, según el tipo de la clase para determinar, asigna el punto a una de ellas.

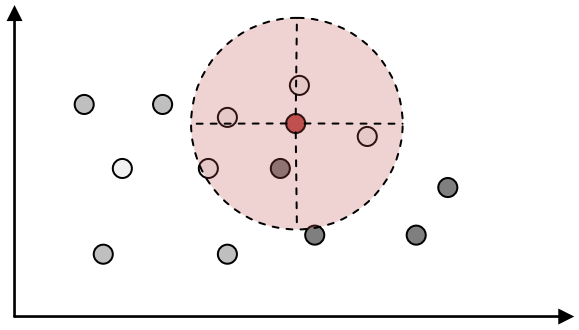

Fig. 1: Ilustración del algoritmo K-NN en dos dimensiones con K=5

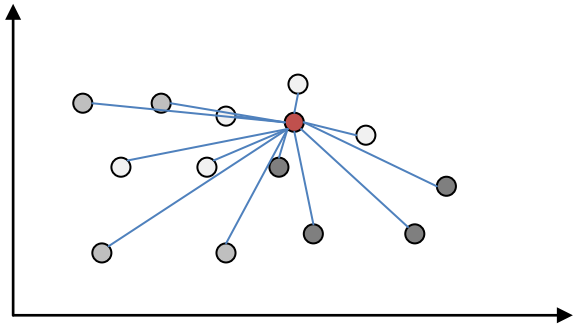

Fig. 2: Distancias entre el punto a clasificar al conjunto de entrenamiento

2.1 Clases nominales
Para clases nominales, se toma la decisión según mayoría. Si no hay ninguna clase prevalente, es posible aumentar el número K hasta alcanzarla, o no tomar ninguna decisión.

2.2 Clases numerales
En el caso de clases numerales, la clase se calcula según las clases de los K vecinos más cercanos. Una función posible es el promedio.

2.3 Análisis del tiempo necesario
Como explicado en la descripción del algoritmo, para una clasificación hay que:

- Determinar las distancias a todos los puntos del conjunto de entrenamiento
- Buscar los vecinos más cercanos (con menor distancia)
- Clasificar según mayoría o por promedio

Para analizar el coste computacional definimos:

- n = número de puntos en el conjunto de entrenamiento
- D = dimensión de los puntos (número de propiedades utilizadas para clasificar)

2.3.1 Distancia entre los puntos
El coste para calcular la distancia entre dos puntos depende de la dimensión de los puntos.

Por ejemplo la distancia euclídea se calcula según la siguiente fórmula:

$$d_e = \sqrt{\sum_i (a_i - x_i)^2}$$

Para D dimensiones eso significa D sustracciones y D multiplicaciones.

Para determinar todas las distancias hay que calcular n distancias, resultando en $n * D$ sustracciones y multiplicaciones o bien:

$$O(n * D)$$

2.3.2 Buscar menores distancias

Para encontrar los K vecinos más cercanos, es necesario ordenar las distancias calculadas. Para hacer esto, se puede utilizar algoritmos rápidos como *Quicksort* con un coste de:

$$O(n * \log n)$$

2.3.3 Clasificar

La clasificación es la parte más sencilla. Para clases nominales se puede buscar la clase prevalente con facilidad, usando un algoritmo con un coste de $O(K)$. Para clases numerales, la función también depende solamente del número de vecinos K resultando en el mismo coste de $O(K)$.

2.3.4 Total

Como resultado el coste total para la clasificación es:

$$O(n * D + n * \log n + K)$$

Normalmente, el número de vecinos tomado en cuenta a la hora de clasificar, K, es relativamente bajo comparado con n. Por eso el término K es insignificante.

Sólo queda $O(n * D + n * \log n)$. Como consecuencia, si la dimensión de los puntos D es mayor que $\log n$, el coste será:

$$O(n * D)$$

Podemos suponer que $D > \log n$ para casi todas las aplicaciones con un elevado número de dimensiones. ($D = 10$ resultaría en $n < 10^{10}$)

3. ACELERAR EL ALGORITMO K-NN

Este trabajo para acelerar el algoritmo de K-NN se centra en aprovechar el hecho de que, si reducimos el número de dimensiones D, podemos acercar el coste a $O(n * \log n)$.

3.1 Reducir la dimensión

Se puede reducir el número de dimensiones con cualquier función que convierte varios datos de entrada en un solo valor que se puede utilizar para calcular la distancia.

Una forma sencilla que cumple este criterio es calcular el promedio.

El factor de reducción R tiene que ser un divisor de D para reducir el número de dimensiones de D a

$$D_i = \frac{D}{R_i}$$

Sin embargo este factor puede variar para cada iteración.

3.2 Clasificar con la dimensión reducida

Con la dimensión reducida no es posible clasificar puntos con la misma calidad que con la dimensión original. Por eso proponemos un algoritmo iterativo:

- (1) Empezar con un factor de reducción alto, para que la dimensión para la primera clasificación sea baja.
- (2) Aplicar el algoritmo K-NN y elegir el porcentaje P más cercano del conjunto de entrenamiento para continuar ($P \gg \frac{K}{n}$)
- (3) "Aumentar" la dimensión de este conjunto reducido, es decir, reducir su dimensión original.
- (4) Continuar (2) & (3) hasta que sólo quedan K vecinos.

3.3 Análisis del tiempo necesario

Suponemos que el coste del algoritmo K-NN utilizado en el algoritmo iterativo se defina como en el capítulo 2: $O(n * D)$

En comparación con el algoritmo sin optimización cambiarán n y D:

- Para la primera clasificación partimos de:
  - o $n_1 = n$
  - o $D_1 \ll$
- Para las siguientes clasificaciones, n_i bajará rápidamente mientras que D_i subirá.
- Aunque el coste del algoritmo K-NN se reduce, se debe tener en cuenta que ahora hay que aplicar dicho algoritmo varias veces y además es necesario calcular la dimensión reducida del punto a clasificar en cada iteración.
- Para obtener una ventaja en el coste computacional es necesario guardar todo el conjunto de entrenamiento con los distintos números de dimensiones utilizados, para que haya que reducir la dimensión de un solo punto para su clasificación. Obviamente eso resulta en un coste de memoria mayor.

3.3.1 Ejemplo

3.3.1.1 Análisis del tiempo

Para demostrar la ventaja del coste computacional suponemos el ejemplo siguiente:

- Reconocimiento / clasificación de imágenes
- Imágenes del tamaño 64x64 píxeles de 8 bit
- Conjunto de entrenamiento de 100.000 imágenes
- Buscar sólo el vecino más cercano ($K = 1$)

De este tamaño se obtiene una dimensión de $D = 4096$, así que en este ejemplo D es mucho más alto que $\log n$.

Con el algoritmo 1-NN original se obtiene los siguientes costes:

- Para calcular todas las distancias: $100.000 * 4096 * 2$ ($* 2$ porque la calculación de la distancia euclídea requiere una sustracción y una multiplicación)

- Para buscar el vecino más cercano: 100.000 (no es necesario ordenar las distancias, sólo hay que buscar la mínima distancia con un coste de $O(n)$)
- En total: $100.000 * 8193 = 819.300.000$

Con el algoritmo utilizando la reducción de dimensiones calculamos lo siguiente:

Para aplicar el algoritmo iterativo hay que fijar los valores de P y R. Valores apropiados para este caso serían, por ejemplo:

- P = 10%, que resulta en:
  - $n_1 = 10^5, n_2 = 10^4, \dots, n_5 = 10$
- R se elige para que en la primera iteración, D_1 sea 1 y en la última, D_5 sea D (4096):
  - $R_1 = 4096$
    $R_2 = 512$
    $R_3 = 64$
    $R_4 = 8$
    $R_5 = 1$
  - o bien: $R_i = 8^{5-i}$
- Las dimensiones que obtenemos para las iteraciones son: $D_i = 8^{i-1}$ o bien:
  - $D_1 = 1$
    $D_2 = 8$
    $D_3 = 64$
    $D_4 = 512$
    $D_5 = 4096$

Para calcular el coste de la clasificación hay que tener en cuenta todas las iteraciones:

- Iteración 1:
  - Coste para calcular las distancias: $100.000 * 1 * 2$
  - Coste para buscar los 10.000 más cercanos para la segunda iteración: $100.000 * 5$ ($\log 100.000 = 5$)
  - En total: $100.000 * 7$
- Iteración 2:
  - Coste para calcular las distancias: $10.000 * 8 * 2$
  - Coste para buscar los 1.000 más cercanos para la siguiente iteración: $10.000 * 4$
  - En total: $10.000 * 20$
- Iteración 3:
  - Coste para calcular las distancias: $1.000 * 64 * 2$
  - Coste para buscar los 100 más cercanos para la siguiente iteración: $1.000 * 3$
  - En total: $1.000 * 131$
- Iteración 4:
  - Coste para calcular las distancias: $100 * 512 * 2$
  - Coste para buscar los 10 más cercanos para la siguiente iteración: $100 * 2$
  - En total: $100 * 1026$
- Iteración 5:
  - Coste para calcular las distancias: $10 * 4096 * 2$
  - Coste para buscar el más cercano: 10
  - En total: $10 * 8193$
- Para las iteraciones 1 a 4 es necesario reducir la dimensión del punto a clasificar: de la reducción mediante el promedio resultan 4096 adiciones y 1, 8, 64 y 512 divisiones que pueden realizarse por desplazamiento binario.
  - En total: $4 * 4096 + 1 + 8 + 64 + 512 = 16.969$
- En total para todas las iteraciones:
  $1.232.499$

Comparado con el 1-NN sin reducción de la dimensión el algoritmo iterativo es aproximadamente 663 veces más rápido.

3.3.1.2 Análisis de la memoria

Obviamente una reducción del coste computacional tan enorme resulta en un aumento de la memoria utilizada.

Para obtener la ventaja temporal del algoritmo, es imprescindible guardar los datos de las dimensiones reducidas de todo el conjunto de entrenamiento para poder calcular las distancias directamente.

En el ejemplo sin optimización del algoritmo K-NN sólo es necesario guardar los datos originales resultando en una utilización de:

- $100.000 * 4096 * 8\ bit \cong 390\ MiB$

Para el algoritmo iterativo hay que guardar, además de estos, los datos para cada iteración:

- $100.000 * 512 * 8\ bit \cong 49\ MiB$
- $100.000 * 64 * 8\ bit \cong 6\ MiB$
- $100.000 * 8 * 8\ bit \cong 1\ MiB$
- En total: $\sim 447\ MiB$

Esto significa que el incremento de 66.374% en la velocidad implica un incremento de un solo 15% en la memoria.

4. APLICACIONES

Debido a la facilidad de visualizar las dimensiones reducidas, el reconocimiento de imágenes es quizá la aplicación más evidente.

En el ejemplo anterior, la imagen original es de 64 por 64 píxeles. Si en lugar de $R_i = 8^{5-i}$ se utiliza $R_i = 4^{5-i}$, cada iteración podría ser visualizada por una imagen del mismo tamaño pero con distintas resoluciones: 32x32, 16x16, 8x8, 4x4.

Sin embargo, en teoría es posible utilizar el algoritmo para cualquier tipo de datos, siempre y cuando sea posible aplicar una función de reducción (como el promedio) a las propiedades. Eso puede limitar el uso con propiedades nominales.

4.1 Ejemplo de representación

En el siguiente ejemplo la imagen tiene un tamaño de 8 por 8 píxeles con tonos de grises de 8 bit. Para aplicar el algoritmo optimizado usamos cuatro iteraciones con las dimensiones siguientes:

- $D_1 = 1 * 1 = 1$
- $D_2 = 2 * 2 = 4$
- $D_3 = 4 * 4 = 16$
- $D_4 = 8 * 8 = 64$

Para calcular los valores de las propiedades reducidas usamos el promedio:

$$M_N = \frac{1}{N}\sum_{i=1}^{N} x_i$$

La imagen original es la siguiente:

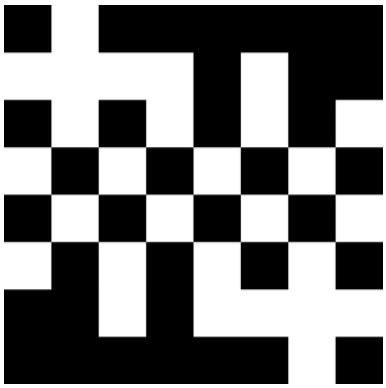

Fig. 3: Imagen original (C_4)

Para demostrar la reducción, conviene empezar con la última iteración, que utiliza la dimensión original, y continuar reduciendo la dimensión.

En D_4 la imagen será la de Fig. 3 con 64 píxeles, cada uno blanco (0) o negro (1).

Para convertir la imagen a D_3 dimensiones o bien píxeles tomamos cuatro píxeles de la imagen original y los unimos a un nuevo pixel de un tono de gris calculado como promedio de los píxeles sustituidos:

- Los cuatro píxeles a la izquierda en la parte de arriba son: (1, 0, 0, 0). El promedio es: $\frac{1}{4}(1 + 0 + 0 + 0) = 0.25$

La imagen resultando de esta conversión se puede ver en Fig. 4.

Fig. 4: Imagen reducida a 16 dimensiones (C_3)

Con el mismo algoritmo se calcula las otras imágenes, cada vez con menos píxeles.

Para $D_1 = 1$ se puede ver que la imagen en Fig. 6 solo consiste de un solo pixel en el tono de gris del promedio de todos los 64 píxeles de la imagen original:

$$C_1(1) = \frac{1}{64}\sum_{i=1}^{64} C_4(i)$$

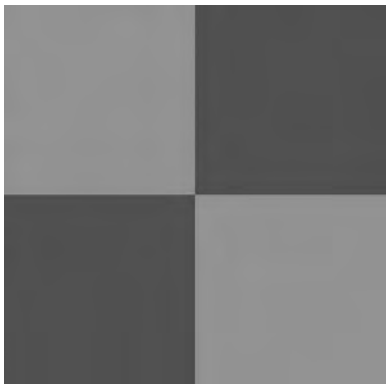

Fig. 5: Imagen reducida a 4 dimensiones (C_2)

Fig. 6: Imagen reducida a una dimensión (C_1)

5. REFERENCIAS

[1] University of Muenster, CVPR Group 2006. Mustererkennung. DOI= http://cvpr.uni-muenster.de/teaching/ws06/mustererkennungWS06/script/ME02-2.pdf

[2] Villena, J. 2009. Inteligencia en Redes de Comunicaciones. Aprendizaje. Universidad Carlos III de Madrid. DOI= http://www.it.uc3m.es/~jvillena/irc/descarga.htm?url=material/06.pdf