Trabalho Prático 01

Rachel Biezuner[1]

[1]Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

rachelbiezuner@gmail.com

Classes

Node

A classe node representa um nó da Árvore KD. Para nós internos, ela guarda as propriedades "dimension" e "plane", que representam respectivamente a dimensão do nó e qual plano ele corta o espaço de pontos. Para nós externos, ela guarda uma lista com os parâmetros do nó.
Além disso, para os nós olhas, existem as variáveis "left" e "right", que guardam dois objetos do tipo "Node", um para o filho esquerdo e o outro para o filho direito.

KDTree

Essa classe representa a Árvore KD. Ela possui um atributo "root" que guarda o nó raiz da árvore. Esse nó raiz guarda seus dois filhos, esquerdo e direito, que guardam seus dois filhos, e assim por diante, até chegar nos nós folha, que não possuem filhos.
Além disso, a classe possui um método "BuildKDTree", que é chamado no construtor. Essa classe retira como parâmetros um dataframe do Pandas "P" e a dimensão, "dimension", que está sendo analisada e cria um nó. Em seguida, se o dataframe tem tamanho maior que 1, ela o ordena em ordem crescente e guarda a "medianPosition", que é a posição da mediana no datarame ordenado. "P" é então dividido em "P1", para seu filho esquerdo, e "P2", para seu filho direito, de acordo com a "medianPosition", e então fazemos duas chamadas recursivas para "BuildKDTree", para encontrar os filhos do nó atual. Na Árvore KD, cada altura da árvore é referente a uma dimensão dos pontos, logo precisamos somar mais um ao valor da dimensão atual e fazer o módulo em relação ao número da dimensão máxima.
Caso o comprimento do dataframe "P" for igual a 1, isso significa que atingimos um nó folha, e basta adicionar esse dataframe, que na verdade é só uma linha, ao nó atual, na forma de uma lista de parâmetros.
A função de recursão do método "BuildKDTree" é $T(n) = 2T(n)/2\Theta(n \log n)$, pois há uma chamada recursiva à função para cada metade do dataframe e é preciso ordenar o dataframe "P", utilizando o algoritmo QuickSort, que tem complexidade de $O(n \log n)$
Utilizando o Teorema Mestre, chegamos na conclusão que a complexidade de tempo do algoritmo é $\Theta(n \log^2 n)$

PriorityQueue

A classe PriorityQueue implementa uma fila de prioridades utilizando uma lista, em vez de heap, pois achei mais prático de implementar. Ela possui um tamanho máximo, que é determinado pelo usuário. Além disso, ela possui métodos para acrescentar pontos, respeitando o tamanho máximo, removê-los, retornar o ponto de maior prioridade e o de

menor prioridade. Como a implementação foi feita em uma lista, todos os métodos têm complexidade $O(n)$.

xNN

A classe "xNN" tem um método "findPriorityQueue" que encontra a fila de prioridades de um ponto de forma recursiva. Nesse caso, a prioridade de um nó folha será sua distância ao ponto, e quanto maior a distância, maior a prioridade.
O algoritmo começa na raiz da Árvore KD, e verifica qual a dimensão do nó. Se o valor do ponto nessa dimensão for maior que o do plano do nó, devemos ir para o nó direito, ou para o esquerdo, caso o contrário. Ou seja, devemos fazer uma chamada recursiva para o método "findPriorityQueue" para o nó escolhido. No entanto, também precisamos avaliar o outro filho. Se a distância do ponto ao plano do outro filho for menor que a maior distância na fila de prioridades, fazemos uma chamada recursiva da função "findPriorityQueue" para o outro nó. Caso o nó atual não tiver filhos, ou seja, for uma folha, devemos tentar adicioná-lo à fila de prioridade. Isso só ocorrerá, é claro, se o nó de maior distância tiver uma prioridade maior que o nó atual.
Um método ingênuo de encontrar o nó com menor distância, seria percorrer a árvore inteira. Não precisamos fazer isso, porque tendo a nossa fila de prioridades, sabemos que não precisamos ir para certos nós se a distância até esse plano for maior que todas as distâncias da fila de prioridades. É por isso que as árvores KD são tão eficientes.
O método "runTest", roda o método "findPriorityQueue" para todos os pontos do conjunto de teste e obtém o vizinho mais próximo da fila de prioridades. Por fim, ele calcula a acurácia no conjunto e a precisão e revocação para cada classe.