TP 01 - Algoritmos Geométricos

Marcela Schuttenberg Polanczyk
matrícula: 2019041582[1]

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

marcelapolanczyk@gmail.com

Implementação:

O código foi feito em python com o auxilio das bibliotecas numpy, pandas e math. Para resolver pequenos bugs de funções da biblioteca padrão e numpy, utilizei funções das bibliotecas sklearn.utils, functools e itertools, que respectivamente foram suados apenas para embaralhar os dados recebidos, trasformar uma lista 2d em 1d e concatenar uma lista e uma array.

Inicialmente utilizei data frames por meio da biblioteca pandas para armazenar os dados lidos nos documentos, contendo, a cada linha, as coordenadas de pontos e as suas classes. Os data frames foram usados na classe kd tree e na classe nó. Em seguuida, para a classe xnn, fiz uso de listas, arrays e dicionarios. Os dicionarios foram essenciais para descobrir a classe de cada ponto, enquanto que as outras duas estruturas foram escolhidas segundo a facilidade de tratar as informaçoes em cada situação.

O código posuui 3 classes. A classe Node, a classe Kd tree e a classe Xnn. A seguir, será explicado os atributos e métodos de cada uma:

1) class Node: A classe Node é utilizada para construir a kd tree. Ela existe com o objetivo de criar um objeto tipo Nó que armazena todas as informações que devem ter em cada vértice da árvore. A classe possui um construtor com 6 atributos: left (aponta para esquerda), right (aponta para direita), data (coordenadas dos pontos caso o nó seja folha e mediana caso o nó seja um nó interno), depth (profundidade em que o nó está na árvore), dim(dimensão no ponto) e classe(classe do ponto).

2) class KdTree: A classe KdTree existe para facilitar a criação da árvore kd. Ela possui um método Build Kd Tree que recebe como atributos o data frame com as informações lidas (coordenadas dos pontos e classes), a profundidade da árvore em que a função se encontra e a dimensão dos pontos. Essa função é recursiva e no final retorna a raiz da árvore de tipo Node. É dividida em dois passos:

Passo base: Caso o dataframe recebido tenha apenas um ponto, este deve ser colocdo em um Node leaf. Um Node leaf possui como informação principal as coordenadas e a classe do ponto, além de apontar para valores nulos. Neste passo, a função retorna o nó folha.

Passo recursivo: Primeiramente os pontos contidos no data frame até o momento são ordenados de acordo com a coordenada "profundidade mod numero de coordenadas", ja que em cada profundidade da árvore é necessário utilizar uma coordenada diferente para o cálculo da mediana. Em segundo lugar, o data frame é dividido ao meio. Em terceiro lugar, é calculado a mediana da "coordenada prufundidade mod numero de coordenada". Em caso de haver numero ímpar de pontos, é escolhido o valor do meio. Em caso de ser número par de coordenadas, é feita a média entre os dois pontos centrais. Em seguida, o valor da mediana é colocado como atributo de nodes, de forma a ter todos as divisões dos planos nos nós internos da kd tree. Por último, a função é chamada recursivamente afim de repetir todo o processo com metade dos dados.

3) x nn: A classe x nn é responsável por construir uma árvore com os dados de treinamento (1), procurar os k pontos dessa árvore mais próximos dos pontos de teste(2), prever qual é a classe dos pontos de teste(3) e informar a acurácia, precisão e revocação(4).

(1) A função buildTrainedKdTree() constrói uma arvore kd com 70% dos dados lidos na entrada simpesmente chamando a função Build_kdTree() na classe Kd tree.

(2) A função searchKdtree(), com auxilio das funções euclidean_distance(), insert() e compare(), retorna uma lista ordenada com os k pontos mais próximos de um ponto teste, contendo suas coordenadas e sua distâncias até esse ponto teste. Para isso, há um caminhamento por todos os nós que não são restritos por uma região certamente com pontos mais afastados. Essa função é recursiva e pode ser separada em duas etapas que serão explicadas a seguir:

Passo base: Verifica-se se já está em uma folha da árvore, ou seja, se os atributos left e right apontam para Nule. Caso for verdade, verifica-se o tamanho da lista de prioridade até o momento. Se ela ainda não estiver cheia, adiciona o ponto na lista e ordena a lista. Caso ela já estiver cheia, faz-se uso da função euclidean_distance() para saber a distância entre o ponto na folha e o ponto teste; em seguida, faz-se uso da função compare() para saber se esse ponto da folha é menhor que algum dos pontos já na fila de prioridade; Caso positivo, faz-se uso da função insert() para inserir essa folha no local correspondente e eliminar a última posição da fila, obtendo-se uma lista atualizada. No fim do passo base, é retornado a fila de prioridade.

Passo recursivo: Caso o vértice for um nó interno, ou seja, não tem seus atributos left e right igual à Nule, é verificado a restrição para caminhar na árvore. Se a coordenada correspondente à coordenada do cálculo da mediana do ponto teste for maior que a mediana no nó atual, então a subárvore da direita tem mais candidatos próximos do ponto teste. A função é chamada recursivamente, portanto, para a subárvore da direita. Se depois dessa chamada recursiva a fila de prioridade ainda não estiver cheia ou ainda há algum ponto na árvore com potencial de estar mais próximo do ponto de teste do que o ponto de menor prioridade na lista, é chamado a função recurivamente para a subárvore da esquerda. Caso o valor a coordenada correspondente à coordenada do cálculo da mediana do ponto teste for menor que a mediana no nó atual, é realizada a mesma lógica, porém considerando que a subárvore da esquerda que possui mais pontos candidatos a

serem os mais próximos.

(3) A previsão de qual é a classe dos pontos testes é realizada no método classe(). Esse método primeiro chama a função searchKdtree(), passando como parâmetro um ponto teste de cada vez e obtendo a lista de prioridade dos pontos mais próximos dele. Essa lista possui em uma de suas sublistas a classe respectiva de cada ponto próximo. Dessa maneira, cria-se um dicionário com essas classes, contendo como chave a classe e como valor a quantidade de vezes que aparece. Após fazer isso com todos os pontos testes, a função retorna uma lista (em que cada posição é um ponto) de arrays que possui a coordenada do ponto e a sua provável classe (a que mais paraceu no dicionário).

(4) A função acuracia() recebe como parametro uma lista de pontos testes, com suas coordenadas, sua verdadeira classe e sua classe suposta pela função classe(). Em seguida, é comparado quantas classes adivinhadas são iguais as verdadeiras e quantas não são. Por último, a acurácia é calculada com o número de acertos/(acertos+erros). Esse código está incompleto e não possui o cálculo de precisão e revocação.

Testes: Não teve tempo de rodar os 10 testes com datas de tamanho diferentes para saber da acurácia geral do código. Apesar disso, a acurácia encontrada para o data set "appendicitis.dat" foi de 0.03125, o de "monk-2.dat" foi 0.47692307692307695 e o de "led7digit.dat" foi de "0.02". Esses valores indicam que ou o método não foi muito eficaz ou há algum erro no cálculo da acurácia.