Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

DCC207 – Algoritmos 2
Prof. Renato Vimieiro

Trabalho Prático 01 – Aplicações de algoritmos geométricos

Nome: Ricardo Furbino Marques do Nascimento
Matrícula: 2020420620

01. Introdução

O objetivo desta documentação é apresentar a implementação do algoritmo de k-Nearest-Neighbors (kNN), em português, k-Vizinhos-Mais-Próximos, através da estrutura de dados Kd-Tree. Serão detalhadas neste relatórios as escolhas feitas na implementação dos algoritmos, incluindo as estruturas de dados, qual foi a forma de implementar e os testes feitos. O trabalho foi desenvolvido na linguagem de programação Python.

02. kNN

A primeira estrutura de dados implementada foi a kNN. Nela, recebemos o número de k vizinhos, uma base de treino e uma de teste. A primeira etapa de sua construção é a montagem de uma Kd-Tree, que será mais detalhada quando estivermos tratando dessa estrutura de dados, através dos dados de treino.

Após esse processo, fazemos a classificação dos dados de teste por meio da Kd-Tree construída. Com a classificação se dá em todos os pontos de teste, a complexidade da classificação é $O(m*(n^{1/2} + k))$, sendo m o número de entradas de teste, um vez que aplicamos m vezes o algoritmo de busca na árvore - será explicado posteriormente. A classificação de cada ponto é tratada na estrutura de dados Kd-Tree.

Por fim, construímos uma matriz de confusão, processo que tem a complexidade 0(n), sendo n todos os pontos da base de teste. Com isso, fica fácil obter as métricas de revocação e precisão para cada label da base de dados, além do acerto geral obtido.

03. Kd-Tree

A estrutura de dados Kd-Tree é necessária assim que a kNN é inicializada. Precisamos dela justamente para construir a lógica de previsão de um ponto. Isso é feito através de uma busca na Kd-Tree, que será posteriormente explicada. Primeiro, é necessário explicar a implementação da construção da árvore Kd.

A construção da Kd-Tree segue o seguinte algoritmo: a partir de uma primeira dimensão arbitrária, ordenamos e, logo após, dividimos a base de treino em duas: uma a esquerda da mediana e outra a direita da mediana. Quando realizamos essa divisão, definimos um nó cabeça, que irá guardar as seguintes informações: valor de divisão da base (mediana), em qual dimensão ocorreu essa divisão, a sub-árvore à esquerda e a sub-árvore à direita. Feito isso, continuamos a divisão da mesma forma, criando novos nós, até chegamos no caso em que temos apenas um ponto na base de dados. Quando isso ocorre, criamos um nó especial, chamado de folha, que guarda as seguintes

informações: a dimensão em que ele foi encontrado, a label que ele possui e as coordenadas desse ponto. Esse processo tem a complexidade O(n lg n), sendo n o número de entradas da base de treino.

Agora que temos explicado esse processo de construção da árvore, partirmos para outro método desta classe: encontrar os k-vizinhos mais próximos de um ponto numa Kd-Tree. O algoritmo recursivo, que é baseado em uma busca em uma árvore, é o seguinte: **a)** caso estamos lidando com um nó que é folha, temos que tentar adicioná-lo na nossa lista de vizinhos (registrada em uma simulação de fila de prioridades) com as seguintes regras – caso a lista de vizinhos ainda não chegou no número k de vizinhos, adicionamos a folha registrando a distância do ponto até o ponto da folha, o ponto da folha e a label da folha; caso a lista de vizinhos esteja cheia, apenas adicionamos o ponto caso a distância o ponto e folha seja menor que a distância do primeiro elemento da lista até o ponto desejado. Para simular a fila de prioridade, teremos que registrar a distância na lista sempre em negativo (maior elemento será o primeiro da lista) e fazer uma ordenação da lista a cada adição de novo ponto. Ou seja, nessa parte do algoritmo, no pior caso, sempre temos a complexidade O(k log k). **b)** Caso estamos lidando com um nó não folha, temos que realizar o seguinte procedimento: caso o valor da mediana seja menor que o valor do ponto na dimensão do corte da mediana, temos que buscar um ponto primeiramente no nó à direita e - caso a lista de vizinhos ainda não esteja completa ou a diferença absoluta entre o valor do ponto na dimensão do corte e a mediana seja maior que o valor po ponto na dimensão de corte do primeiro elemento da lista de vizinhos - precisamos fazer a busca também no nó da esquerda. Caso o valor da mediana seja maior ou igual que o valor do ponto na dimensão do corte da mediana, temos o caso inverso: primeiro varremos o lado esquerdo e, caso necessário, varemos o lado direito. Dada a complexidade explicada anteriormente e o algoritmo do passo b, o algoritmo tem complexidade $O(n^{1/2} + k)$.

04. Testes

Foram utilizadas 10 bases de dados diferentes para o teste da classe kNN e, consequentemente, a classe Kd-Tree. Foi uma pequena adequação dos dados à forma na qual o algoritmo foi implementado: a coluna que possui a classificação de cada ponto foi transposta para a primeira posição. Após esse tratamento, foram feitos dois testes para cada base: k=3 e k=5. Há uma grande variância nas métricas atingidas em cada teste, entretanto, em todos foi possível observar que com mais vizinhos (k=5) o algoritmo se tornava mais preciso. No arquivo main.py, temos todas os resultados impressos no terminal, inclusive com a classificação completa de cada base de teste.