Documentação Trabalho Prático -1
Algoritmos 2

Samuel Lipovetsky

Departamento de Ciência da Computação - Universidade Federal de Minas Gerais(UFMG)
Belo Horizonte - MG - Brasil
samuellpvt@gmail.com

1 Árvore Kd

1.1 Explicação

A árvore kd foi implementada a partir de uma classe que tem como construtor um método que recebe uma lista de pontos. Em seguida, é construída uma árvore binária recursivamente separando cada dimensão pela mediana , os pontos menores que a mediana ficam a esquerda do nó pai e o restante a direita . Dessa forma , todo nó que é uma folha representa um ponto e todo nó que não é uma folha representa um hiperplano que divide uma dimensão em dois espaços.

1.2 Implementação

A parte mais custosa no processo de construir uma árvore Kd é encontrar a mediana que divide uma dimensão em toda chamada recursiva , por isso , foi implementado o método quickSelect que retorna a mediana de uma lista de elementos com complexidade de tempo $\theta(n)$ para o caso médio. Essa função particiona uma lista original em três listas lt, eq, gt de elementos menores, iguas e maiores do que um pivô selecionado, em seguida é checada o tamanho das listas criadas. Seja i o i-ésimo menor elemento da lista original, temos :

$$\begin{cases} i \leq |lt| \rightarrow i \in lt \\ |lt| < i \leq |lt| + |eq| \rightarrow i \in |eq| \\ i > |lt| + |eq| \rightarrow i \in gt \end{cases} \tag{1}$$

Portanto, apos particionar a lista original sabemos a qual partição o i-ésimo menor elemento pertence e recursivamente é reduzido o tamanho da lista até existir somente um elemento ou $i \in |eq|$, pois como todos os elementos de eq são iguais se i está em eq o i-ésimo menor elemento tem o valor de qualquer elemento de eq. Dessa forma, achar a mediana é somente um caso especial de achar o i-ésimo menor elemento em que i é a metade do tamanho da lista original.

Após encontrar a mediana que divide uma dimensão é construído um objeto com a Classe Node que pode representar tanto um hiperplano em nós intermediários como pontos nas folhas. A árvore kd é construída recursivamente e a cada chamada recursiva o número da dimensão que é divida e incrementada em 1 .

2 K Vizinhos mais próximos

2.1 Explicação

O Knn foi implementado com uma Classe que recebe de argumentos um conjunto de pontos que serão utilizados como conjunto de treinamento, mais especificamente será construída uma árvore kd utilizando a classe já descrita.

Ainda dentro da classe Knn , temos o método knear que é de fato responsável por executar o algoritmo. Inicialmente é recebido como argumento a raiz de uma árvore kd , um ponto de teste e o número de vizinhos próximos. Em seguida , o algoritmo busca por candidatos a vizinhos mais próximos com uma recursão até uma folha,porém, em vez de simplesmente retornar essa folha é feito um teste durante a subida das chamadas recursivas que garante que os pontos escolhidos são os mais próximos. Esse teste é feito usando a maior distância X entre o ponto de teste e um ponto candidato. Se um hiperplano intercepta a esfera descrita pelo ponto de teste e a distância X então é necessário testar ambos os lados desse hiperplano, caso contrário, é preciso testar para k vizinhos mais próximos somente o lado do hiperplano em que o ponto de teste existe.

2.2 implementação

Para armazenar os K vizinhos mais próximos foi utilizada uma Heapq que armazena objetos da classe Node. Com fim de garantir funcionamento da heapq a classe node tem um método __lt__ que é usado como comparador e utilizada as distâncias entre o ponto de teste e os candidatos a vizinhos mais próximo como critério de ordenação. O heapq garante que o primeiro elemento da heap sempre será o menor , porém é necessário saber a maior distância entre o ponto de teste e os vizinhos mais próximos,então todos os valores dentro da heap são multiplicados por -1 para que o primeiro elemento da heap seja o maior em modulo.

Dessa forma, durante a descida da arvore kd quando uma folha é e encontrada , se a heap não estiver cheia o ponto da folha é inserido na heap , se a heap estiver cheia então é conferido se a distância entre o ponto encontrado e o ponto de teste é menor que a maior distância na heap , se isso for verdade então é removido da heap esse ponto com a maior distância e inserido na heap o novo ponto encontrado.

Após a execução desse algoritmo teremos na heap um conjunto de pontos que serão os k vizinhos mais próximos de um ponto de teste. Dessa forma , podemos conferir a classe a qual esses k vizinhos pertencem e decidir por maioria a classificação do ponto de teste.

3 Gerando estatísticas

Após a construção da árvore kd com o conjunto de treinamento ,o processo de achar os k vizinhos e classificar um ponto de acordo com a maioria é feito para todos os pontos do conjunto de teste, assim é gerado um dicionário de pontos que explicita a classificação a partir dos k vizinhos e a qual classe aquele ponto realmente pertence .Em seguida é possível gerar as estatísticas para o modelo criado como acurácia ,precisão e revocação contando os falso positivos , falso negativos, verdadeiro positivos e verdadeiro negativos

Foi feito também um plot para cada dataset variando os k vizinhos para demonstrar como esse parâmetro altera a precisão , acurácia e revocação do modelo. Todos os plots podem ser encontrados no diretório figures/stats

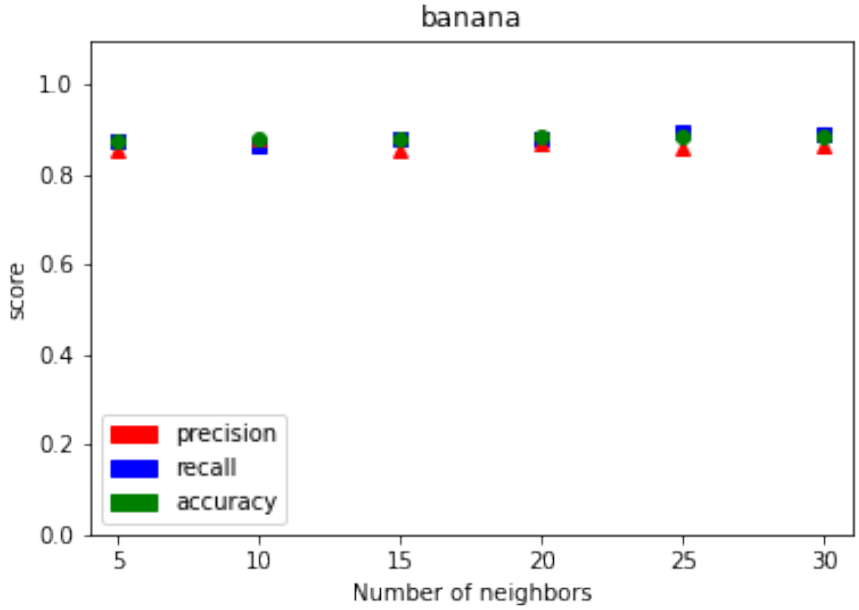
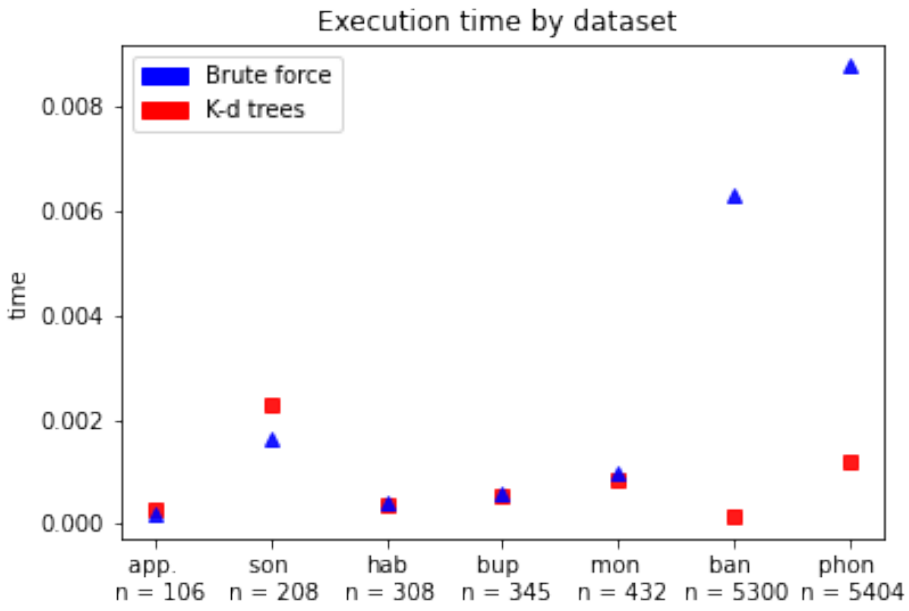


Figura 1: Exemplo de plot gerado

4 Exploração dos dados e testes

Para garantir tanto a corretude dos algoritmos implementados quanto o tempo de execução foi realizado alguns testes que comparam o resultado e o tempo do método knn a partir de árvores kd com o método de força bruta.

4.1 Árvores kd e força bruta



Para todos os datasets utilizados o método knn implementado a partir de árvores kd retornou o mesmo resultado que o método de força bruta.Para datasets grandes como banana e phoneme o método que utiliza as árvores kd foi significativamente mais rápido que o método de força bruta, porém para datasets menores a diferença no tempo de execução foi pequena.

5 Referências

Quicksort Algorithm - Wikipédia , 2021. Disponível em https://pt.wikipedia.org/wiki/Quicksort

Computational Geometry : Algorithms and Applications , 3^{rd} De Berg et al

Vimieiro , R. (2021). Slides virtuais da disciplina de Algoritmos 2 Disponibilizado via Teams - Grupo Algoritmos 2 2021/2