Python KNeighborsClassifier Algorithm

Asafe Clemente Gadelha de Medeiros
asafecgm1@gmail.com
DCC - UFMG
Belo Horizonte, Brasil

ABSTRACT

Um algoritmo para classificação baseado nos k vizinhos mais próximos (kNN) para predizer os rótulos de instâncias de um conjunto de teste, através da coleção do conjunto de treino. Através de árvores kd, podemos buscar pontos próximos com um custo baixo.

KEYWORDS

KNeighborsClassifier, classificação, kd-tree, datasets

1 INTRODUÇÃO

No problema de classificação, uma coleção de instâncias rotuladas, chamado de conjunto de treinamento, é usada para treinar um modelo, o qual é usado para predizer o valor de novas instâncias.

O kNN atribui um rótulo a uma nova instância com base nos seus k vizinhos mais próximos no conjunto de treinamento. O algoritmo considera que as instâncias são pontos no espaço n-dimensional e que a proximidade entre elas é a distância euclidiana.

A qualidade do modelo é avaliada no conjunto de teste, usado para predizer o valor das instâncias, dessa forma, métricas de avaliação medem a qualidade do aprendizado.

Na sessão X é discutido a complexidade tempo de processamento do algoritmo de knn implementado, baseado na árvore kd.

1.1 Objetivo

O trabalho consistiu em criar um algoritmo de classificação knn através de uma árvore k-d para prever as classes de 10 conjuntos Keel-datasets.

1.2 Apresentação geral

O projeto foi feito utilizando Python 3 e as bibliotecas: numpy, pandas e heapq para facilitar na armazenagem e manipulação dos dados. O projeto tem a seguinte estrutura:

- project
  - dataset
  - results
  - main.py
  - knn.py
  - kdtree.py
  - utils.py

Os código principal a ser executado está no main.py, que utiliza das bases de dados no diretório dataset. Em "kdtree.py" é definida a classe KDNode, responsável por guardar um nó da árvore, e um método para criar uma árvore a partir de um conjunto de pontos. O arquivo "knn.py" contém a definição da classe utilizada para receber um conjunto de pontos treino e outro de teste, construir a árvore kd correspondente, e, também, classificar o conjunto de teste. Em "utils.py" encontram-se métodos auxiliares e em results todos os resultados obtidos.

1.3 Estruturas Utilizadas

Inicialmente cada base de dados '.dat' foi lida, criado um dataframe com os dados do arquivo, foi dividido em dois conjuntos, treino e teste na proporção de 30% para teste. Um objeto x_NN é criado, e tem as métricas precisão, revocação e acurácia calculadas.

2 K-D TREE

Na fase de treinamento, temos que construir a árvore kd. Nessa árvore cada nível divide o espaço n-dimensional em uma dimensão (n) e permite uma busca mais rápida pelos pontos mais próximos. Para isso, o ponto que é a mediana tem o valor na dimensão (n) tomado como base e filhos, da esquerda, são menores ou iguais ao esse valor chave, enquanto os que ficam do outro lado, nós da direita, pontos maiores nessa dimensão.

3 CLASSIFICAÇÃO

A etapa de classificação, centro do processo foi feita além da abordagem simples, que seria buscar onde o ponto alvo se encaixa na árvore. Também foi levado em conta que caso a distância até algum eixo seja menor que até o vizinho mais longe podem existir vizinhos no outro ramo, ainda não explorado. Para isso, retrocedemos conferindo essa distância em cada nó superior e se for o caso explorar o outro ramo em busca de vizinhos. O seguinte algoritmo que faz esse trabalho, note que k_best verifica se o nó encontrado é um vizinho mais próximo, armazenados na fila de prioridade (heap) pela distância até o ponto alvo.

Algorithm 1 Procura os vizinhos mais próximos de um ponto através de uma kd-tree

Require: $node, target\_point, neighbors, number\_neighbors(k)$
Ensure: $neighbors$
  procedure KNN ($node, neighbors$)
    if $node$ is a leaf **then**
      $neighbors \leftarrow k\_best\ (neighbors \cup node.point)$
      return $neighbors$
    $next \leftarrow None$; $outro\_no \leftarrow None$
    if $target\_point[node.current\_axis] <= node.key$ **then**
      $next \leftarrow node.left$
      $outro\_no \leftarrow node.right$
    else
      $next \leftarrow node.left$
      $ooutro\_no \leftarrow node.right$
    $neighbors \leftarrow KNN(next, neighbors)$
    if $(neighbors[0] >= dist)$ *or* $(len(neighbors) < k)$ **then**
      $neighbors = KNN(outro\_no, neighbors)$
    return $neighbors$

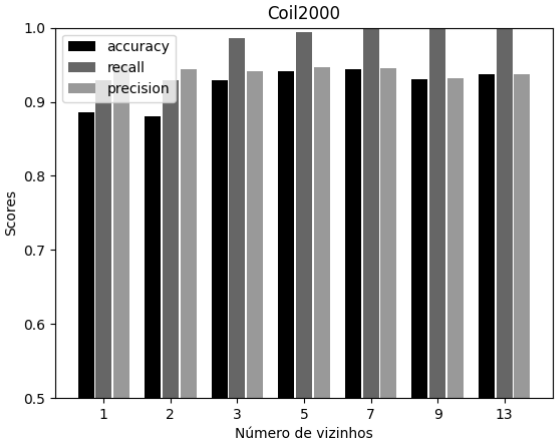

Figure 2: Resultados para a base coil2000.dat

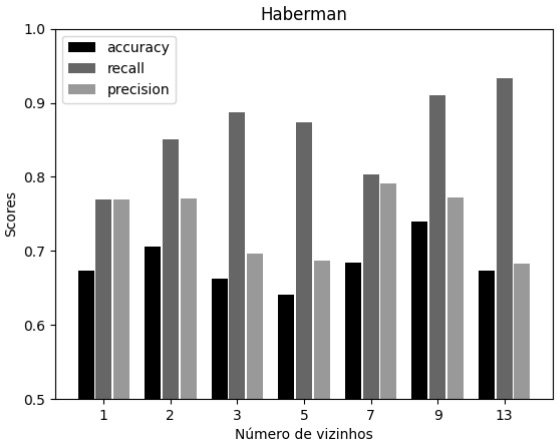

Figure 3: Resultados para a base haberman.dat

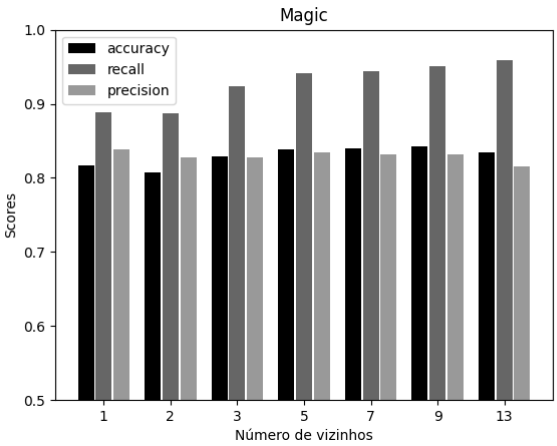

Figure 4: Resultados para a base magic.dat

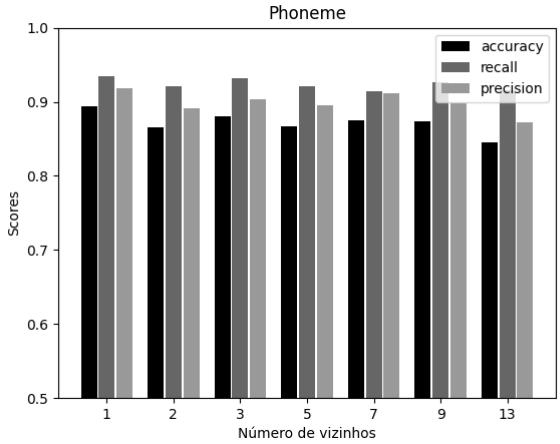

Figure 5: Resultados para a base phoneme.dat

Uma observação relevante é que caso não exista nenhuma similaridade entre o item alvo e itens que o usuário possa já ter avaliado no passado o "score" é computado considerando tanto a média das notas que usuário atribuiu quanto a média das avaliações recebidas pelo item alvo.

4 COMPLEXIDADE COMPUTACIONAL

Na construção da árvore, a etapa mais cara é encontrar a mediana, processo feito através da ordenação dos pontos, que tem custo $n(logn)$ para a quantidade de pontos no conjunto de treinamento. Na etapa de previsão, a estrutura da árvore kd faz operação de consulta potencial vizinho mais próximo em tempo $log(n)$ por se tratar de uma árvore balanceada. Como temos também a condição da distancia até o eixo ser maior que a distancia até os vizinhos, podemos ter que buscar vizinhos em T nós, sendo T a altura da árvore. Entretanto, conforme outros ramos vão sendo explorados a essa distancia até o eixo é cada vez maior, diminuindo essas vezes. No geral podemos considerar que para k vizinhos a complexidade total é $O(k * log(n))$.

5 RESULTADOS

Os resultados, precisão, revocação e acurácia de cada uma das bases podem ser visto nas figuras de 1 a 10. Para cada uma delas os resultados para os números de vizinhos 1,2,3,5,7,9,13 foram gerados e são bastantes satisfatórios, podem variar de acordo com a divisão treino e teste feita, mas geral um número de vizinhos intermediário se mostrou a melhor escolha.

6 CONCLUSÃO

Utilizar uma técnica de geometria computacional, as árvores kd para procurar os pontos mais próximos se mostrou eficaz e possibilitou buscar os vizinhos assim implementando o algoritmo de KNN, fácil e útil em aprendizado de máquina facilmente.

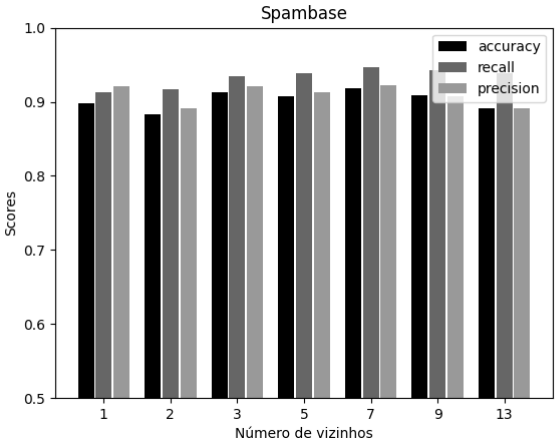

Figure 8: Resultados para a base spambase.dat

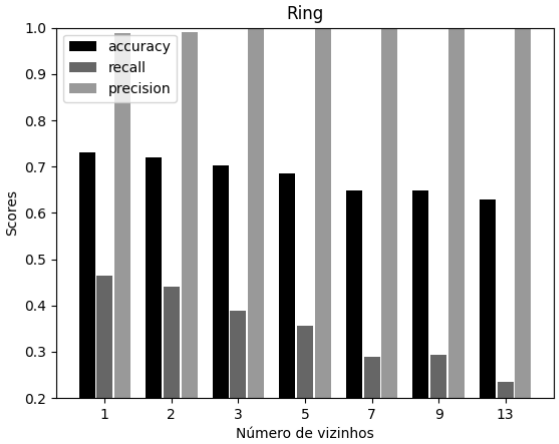

Figure 7: Resultados para a base ring.dat

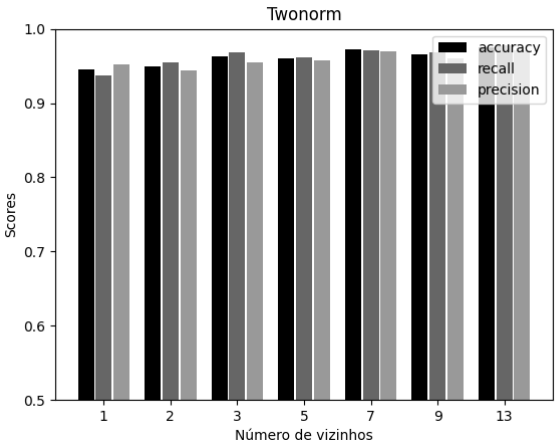

Figure 9: Resultados para a base twonorm.dat

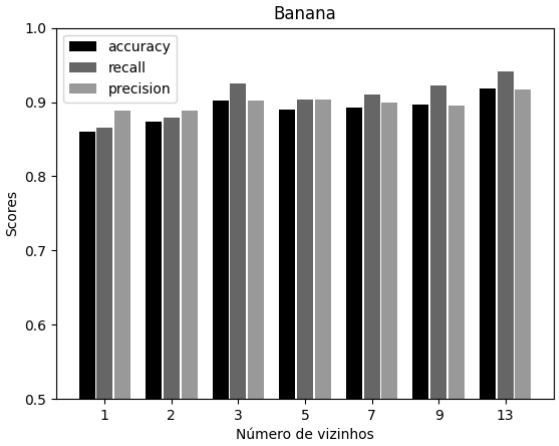

Figure 1: Resultados para a base banana.dat

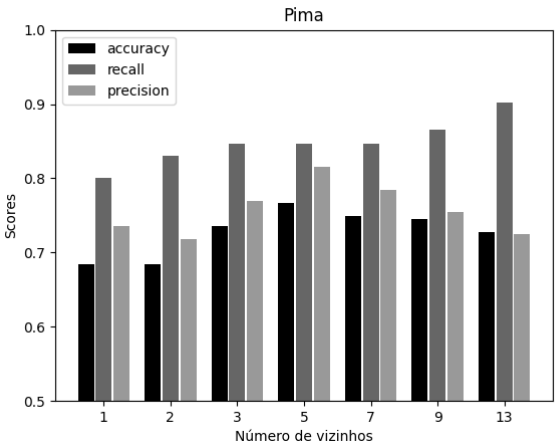

Figure 6: Resultados para a base pima.dat