Implementação do Método de Classificação k-NN

Sávio de Oliveira Camacam1, Henrique Souza Pinheiro1

1Departamento de Computação – Universidade Tecnológica Federal do Paraná (UTFPR)  
87301-899 – Campo Mourão – PR – Brasil

saviocamacam@alunos.utfpr.edu.br, henriquepinheiro@alunos.utfpr.edu.br

**Abstract.** This meta-article describes the process of implementation of the K-Nearest Neighbors ranking algorithm, and the resulting values ​​of the experiments provided as a base.

**Resumo.** Este meta-artigo descreve o processo de implementação do algoritmo de classificação K-Nearest Neighbors, bem como os valores resultantes dos experimentos fornecidos como base.

# 1. Informações Gerais

Em reconhecimento de padrões, o algoritmo *key-Neares Neighbors* (ou *k-NN*) é um método não paramétrico usado para classificação, onde a entrada consiste em um número de exemplos mais próximos no espaço de características, e a saída nesse caso é a sugestão de uma classe, onde um objeto testado é classificado pela identificação da classe com mais exemplares nessa entrada.

Em suma, há um conjunto de valores de treinamento e outro conjunto de testes, onde dada uma proporção dos valores de treino, é calculada a distância de cada valor de teste com todos os valores dessa proporção a fim de eleger as *k* menores distâncias e partir daí eleger uma generalização de classe para o objeto testado[Wiki].

Esse trabalho trata da implementação desse algoritmo empregando uma heurística para cálculo da aproximação das características entre as classes do espaço de treinamento e o objeto testado, que seja a Distância Euclidiana, de Manhattan ou outra.

O conjunto de dados para treinamento é um arquivo de 3600 linhas com 24 colunas que representam as características, divididas em 12 classes que são os meses do ano. O arquivo de teste tem os mesmos parâmetros em 1200 linhas. Para essa implementação esses valores são invisíveis suportando qualquer quantidade nesses parâmetros.

# 2. Estruturas e Procedimentos

A implementação foi feita usando Java. Foram elencados os seguintes objetos:

* Característica: guarda um valor de avaliação e identifica as colunas do arquivo e guarda os menores e maiores valores daquela característica;
* Classe: agrupa as linhas de uma determinada classe;
* Linha: é um agrupamento das características lidas no arquivo, tem a informação da classe correta e um campo de sugestão de classe, e um valor de distância;

# 2.1. Normalização dos Valores

O método de normalização é executado para o conjunto de linhas de cada arquivo do tipo Linha e com dois *for* aninhados correndo cada Linha de cada arquivo dentro do for mais interno é indicado os valores normalizados para um conjunto de características organizado em função de colunas.

A partir daí são encontrados os valores mínimos e máximos de cada característica para aplicação do método de normalização *min-max* escolhido para esse trabalho segundo a fórmula:

# 2.2. Sorteio dos Valores para treino

Esse método foi implementado primeiro em termos da validação cruzada. O objeto responsável guarda em sua estrutura duas listas de objetos do tipo Linha, sendo que esta guarda valores do tipo Característica, nesse caso, as 24 passadas como entrada, além de uma informação da classe a que pertence e um campo de sugestão de classe.

No caso de *cross-validation* é feita a seleção das linhas a partir de um conjunto do tamanho da proporção passada por parâmetro na execução do arquivo. Essa seleção vai da separação de todas as linhas de acordo sua classe, e ao final, independente da tarefa, sendo cross-validation ou o teste propriamente dito, as linhas sorteadas estarão a disposição numa lista chamada *conjuntoTreino.*

# 2. Validação Cruzada

Em meio a interpretações deturpadas do trabalho proposto, foi implementado um método de validação cruzada, onde aplicada uma proporção os valores de treino num dado arquivo, o programa executa o teste com a outra parte desse mesmo arquivo respeitando as classes correspondentes das entradas.

Essa implementação não foi requerida, mas foi interessante porque de uma forma ou outra o comportamento do algoritmo é exatamente o mesmo e não representou nenhum prejuízo.

# 3. Variação da Proporção

É solicitado a avaliação do impacto de várias proporções do treino afim de notar um padrão nos acertos. As proporções solicitadas são 25%, 50% e 100%, mas o programa aceita o cálculo de qualquer proporção.

O desempenho é dado nas tabelas rotuladas a seguir:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **25% com k=3** | | | | | | **Resultado:** 62.3336 | | | | | | **%** |
| **49.0** | 23.0 | 1.0 | 1.0 | 7.0 | 11.0 | 2.0 | 0.0 | 3.0 | 1.0 | 1.0 | 1.0 | **49.0** |
| 17.0 | **67.0** | 1.0 | 0.0 | 3.0 | 4.0 | 0.0 | 1.0 | 1.0 | 0.0 | 4.0 | 2.0 | **67.0** |
| 2.0 | 2.0 | **75.0** | 1.0 | 6.0 | 2.0 | 1.0 | 3.0 | 3.0 | 2.0 | 2.0 | 1.0 | **75.0** |
| 2.0 | 1.0 | 6.0 | **73.0** | 10.0 | 0.0 | 0.0 | 2.0 | 1.0 | 4.0 | 0.0 | 1.0 | **73.0** |
| 6.0 | 11.0 | 4.0 | 9.0 | **40.0** | 1.0 | 1.0 | 2.0 | 7.0 | 6.0 | 10.0 | 3.0 | **40.0** |
| 12.0 | 6.0 | 0.0 | 2.0 | 2.0 | **55.0** | 7.0 | 1.0 | 3.0 | 0.0 | 11.0 | 1.0 | **55.0** |
| 4.0 | 2.0 | 1.0 | 0.0 | 4.0 | 22.0 | **57.0** | 1.0 | 0.0 | 5.0 | 4.0 | 0.0 | **57.0** |
| 1.0 | 1.0 | 2.0 | 3.0 | 2.0 | 0.0 | 1.0 | **72.0** | 1.0 | 1.0 | 1.0 | 15.0 | **72.0** |
| 1.0 | 1.0 | 1.0 | 4.0 | 2.0 | 1.0 | 2.0 | 0.0 | **74.0** | 9.0 | 5.0 | 0.0 | **74.0** |
| 0.0 | 7.0 | 2.0 | 1.0 | 6.0 | 2.0 | 3.0 | 0.0 | 11.0 | **67.0** | 1.0 | 0.0 | **67.0** |
| 2.0 | 4.0 | 6.0 | 1.0 | 21.0 | 6.0 | 1.0 | 2.0 | 2.0 | 8.0 | **44.0** | 3.0 | **44.0** |
| 1.0 | 2.0 | 2.0 | 0.0 | 5.0 | 0.0 | 0.0 | 4.0 | 6.0 | 3.0 | 2.0 | **75.0** | **75.0** |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **50% com k=3** | | | | | | **Resultado:** 65.3333 | | | | | | **%** |
| **48.0** | 27.0 | 0.0 | 5.0 | 6.0 | 5.0 | 3.0 | 0.0 | 1.0 | 1.0 | 2.0 | 2.0 | **48.0** |
| 14.0 | **70.0** | 3.0 | 3.0 | 1.0 | 2.0 | 0.0 | 2.0 | 0.0 | 1.0 | 2.0 | 2.0 | **70.0** |
| 1.0 | 4.0 | **76.0** | 5.0 | 4.0 | 0.0 | 0.0 | 0.0 | 3.0 | 1.0 | 3.0 | 3.0 | **76.0** |
| 3.0 | 0.0 | 5.0 | **81.0** | 3.0 | 0.0 | 0.0 | 1.0 | 0.0 | 5.0 | 1.0 | 1.0 | **81.0** |
| 4.0 | 12.0 | 2.0 | 11.0 | **45.0** | 1.0 | 2.0 | 2.0 | 5.0 | 4.0 | 11.0 | 1.0 | **45.0** |
| 10.0 | 6.0 | 0.0 | 1.0 | 1.0 | **64.0** | 6.0 | 2.0 | 3.0 | 1.0 | 6.0 | 0.0 | **64.0** |
| 5.0 | 3.0 | 1.0 | 0.0 | 2.0 | 22.0 | **55.0** | 2.0 | 2.0 | 6.0 | 1.0 | 1.0 | **55.0** |
| 2.0 | 0.0 | 1.0 | 2.0 | 1.0 | 0.0 | 0.0 | **79.0** | 1.0 | 1.0 | 0.0 | 13.0 | **79.0** |
| 2.0 | 2.0 | 0.0 | 4.0 | 3.0 | 1.0 | 1.0 | 0.0 | **75.0** | 5.0 | 6.0 | 1.0 | **75.0** |
| 2.0 | 4.0 | 3.0 | 0.0 | 5.0 | 1.0 | 4.0 | 0.0 | 8.0 | **71.0** | 2.0 | 0.0 | **71.0** |
| 3.0 | 5.0 | 5.0 | 2.0 | 6.0 | 6.0 | 0.0 | 2.0 | 6.0 | 10.0 | **50.0** | 5.0 | **50.0** |
| 2.0 | 1.0 | 2.0 | 1.0 | 7.0 | 0.0 | 1.0 | 8.0 | 6.0 | 0.0 | 2.0 | **70.0** | **70.0** |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **100% com k=3** | | | | | | **Resultado:** 67.0 | | | | | | **%** |
| **57.0** | 24.0 | 1.0 | 2.0 | 5.0 | 4.0 | 0.0 | 0.0 | 2.0 | 2.0 | 1.0 | 2.0 | **57.0** |
| 15.0 | **67.0** | 2.0 | 3.0 | 1.0 | 3.0 | 0.0 | 2.0 | 1.0 | 2.0 | 2.0 | 2.0 | **67.0** |
| 1.0 | 1.0 | **77.0** | 3.0 | 7.0 | 1.0 | 1.0 | 0.0 | 1.0 | 3.0 | 4.0 | 1.0 | **77.0** |
| 2.0 | 0.0 | 4.0 | **85.0** | 2.0 | 0.0 | 0.0 | 0.0 | 0.0 | 4.0 | 0.0 | 3.0 | **85.0** |
| 5.0 | 8.0 | 2.0 | 11.0 | **50.0** | 1.0 | 2.0 | 3.0 | 7.0 | 3.0 | 7.0 | 1.0 | **50.0** |
| 16.0 | 4.0 | 0.0 | 1.0 | 2.0 | **62.0** | 3.0 | 1.0 | 1.0 | 1.0 | 8.0 | 1.0 | **62.0** |
| 4.0 | 1.0 | 0.0 | 0.0 | 1.0 | 21.0 | **60.0** | 2.0 | 1.0 | 4.0 | 5.0 | 1.0 | **60.0** |
| 2.0 | 0.0 | 1.0 | 3.0 | 3.0 | 0.0 | 1.0 | **81.0** | 0.0 | 0.0 | 0.0 | 9.0 | **81.0** |
| 1.0 | 1.0 | 2.0 | 1.0 | 5.0 | 4.0 | 2.0 | 0.0 | **73.0** | 7.0 | 4.0 | 0.0 | **73.0** |
| 1.0 | 8.0 | 2.0 | 0.0 | 2.0 | 0.0 | 3.0 | 0.0 | 11.0 | **71.0** | 2.0 | 0.0 | **71.0** |
| 2.0 | 4.0 | 5.0 | 2.0 | 7.0 | 6.0 | 0.0 | 2.0 | 4.0 | 9.0 | **50.0** | 9.0 | **50.0** |
| 1.0 | 2.0 | 1.0 | 3.0 | 4.0 | 0.0 | 1.0 | 9.0 | 5.0 | 1.0 | 2.0 | **71.0** | **71.0** |

# 4. Variação dos K-Vizinhos Mais Próximos

Em situações de variações para mais no *k* foi notada alternância nos erros e acertos variando um pouco mais que 2% da taxa atual, permanecendo a mesma taxa ou diminuindo em até 2%.

# 5. Tempo de Execução e Performance

Em observação da execução do programa, as buscas ocorreram em média a partir de 20s chegando a 40s com teste de proporção de 100% com *k* igual a 3.

# 6. Conclusão

Em primeiro momento houve confusão na interpretação do problema, mas o trabalho contribuiu de forma significativa na criação da percepção de métodos de classificação.

# 7. Referências

[Wiki] k-nearest neighbors algorithm disponível em <https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm>