



# Trabalho de programação II

Classificação de dados utilizando um KNN

André Pacheco e Jordana Salamon

## TRABALHO DE PROGRAMAÇÃO II

DEPARTAMENTO DE INFORMÁTICA - UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Este documento descreve o trabalho da disciplina de programação II 2018/2 ofertada pelo departamento de informática para o curso de bacharelado em Ciência da Computação.

*05 de outubro de 2018*



## Sumário

<b>1</b>	<b>Introdução</b>	<b>4</b>
1.1	Motivação e objetivo	4
<b>2</b>	<b>Descrição do trabalho</b>	<b>5</b>
2.1	Classificação de dados	5
2.2	K-vizinhos mais próximos	6
2.3	Métricas de desempenho de classificadores	7
2.3.1	Acurácia	7
2.3.2	Matriz de confusão	8
2.4	Detalhes da implementação	8
2.5	Padrão de entrada	8
2.6	Padrão de saída	10
<b>2.7</b>	<b>Regras gerais</b>	<b>10</b>
2.7.1	Nota e critério de avaliação	11
2.7.2	Submissão	11
2.7.3	Dúvidas ou problemas encontrados na descrição do trabalho	12
<b>2.8</b>	<b>Fontes de referências</b>	<b>12</b>



# 1. Introdução

## 1.1 Motivação e objetivo

A classificação de dados está presente em diversos problemas reais, tais como: reconhecer padrões em imagens, diferenciar espécies de plantas, classificar tumores benignos e malignos, dentre outros. Este problema é um dos tópicos mais ativos na área de aprendizado de máquina (*machine learning*). Portanto, este será o tema do nosso trabalho de programação.

Em resumo, a ideia principal deste trabalho é implementar um algoritmo de classificação de dados utilizando a linguagem C. Um algoritmo de classificação é capaz, a partir de um conjunto de regras, de "aprender" um padrão de dados a fim de atribuir rótulos a novas amostras. Existem diversos algoritmos, como redes neurais, árvores de decisão, máquinas de vetores de suporte, dentre muitos outros. Alguns deles são estocásticos, ou seja, o algoritmo utiliza meios estatísticos em sua implementação, e outros são determinísticos. Para este trabalho foi escolhido o algoritmo K-vizinhos mais próximos (KNN - *K-Nearest Neighbors*). O KNN é um método determinístico, muito famoso e utilizado no mundo todo.

## 2. Descrição do trabalho

### 2.1 Classificação de dados

Basicamente, o problema de classificação consiste em determinar o rótulo de algum objeto, baseado em um conjunto de atributos extraídos do mesmo. Para que isso ocorra é necessário um conjunto de treinamento com instâncias na qual os rótulos os objetos são conhecidos. Devido a isso, na terminologia de aprendizado de máquina, a classificação de dados é um problema de aprendizado supervisionado.

Um exemplo clássico de classificação é o problema da Iris<sup>1</sup>. Neste problema, existe um conjunto de flores do gênero Iris que são divididas em 3 grupos (rótulos): setosa, virginica e versicolor. Sendo assim, o objetivo é determinar a qual grupo uma determinada flor pertence baseado nas medidas de sépalas e pétalas das mesmas. A Figura 2.1 ilustra um processo de classificação. Inicialmente as sépalas e pétalas devem ser extraídas em um pré-processamento. As medidas extraídas são processadas e suas características extraídas. Por fim, é realizada a classificação das flores. Neste exemplo os valores de entrada (também conhecido com *features*) serão as medidas de comprimento e largura das sépalas e pétalas e os rótulos (também conhecido como *labels*) assumirão os rótulos setosa, virginica e versicolor.

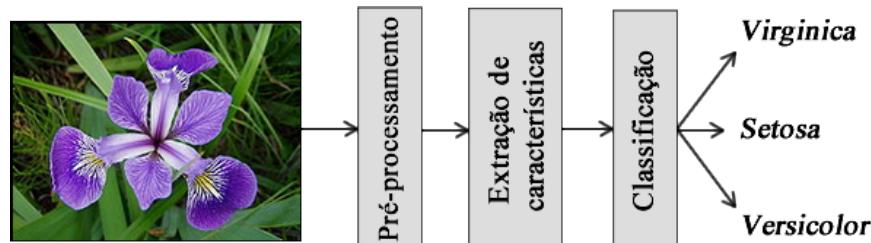


Figure 2.1: Exemplo do processo de classificação de flores do gênero

<sup>1</sup>[https://en.wikipedia.org/wiki/Iris\\_flower\\_data\\_set](https://en.wikipedia.org/wiki/Iris_flower_data_set)

## 2.2 K-vizinhos mais próximos

O algoritmo do K-vizinhos mais próximos (KNN) foi proposto por Fukunaga e Narendra em 1975<sup>2</sup>. É um dos classificadores mais simples de ser implementado, de fácil compreensão e ainda hoje pode obter bons resultados dependendo de sua aplicação.

A ideia principal do KNN é determinar o rótulo de classificação de uma amostra baseado nas amostras vizinhas advindas de um conjunto de treinamento. Nada melhor do que um exemplo para explicar o funcionamento do algoritmo como o da Figura 2.2, na qual temos um problema de classificação com dois rótulos de classe e com  $k = 7$ . No exemplo, são aferidas as distâncias de uma nova amostra, representada por uma estrela, às demais amostras de treinamento, representadas pelas bolinhas azuis e amarelas. A variável  $k$  representa a quantidade de vizinhos mais próximos que serão utilizados para averiguar de qual classe a nova amostra pertence. Com isso, das sete amostras de treinamento mais próximas da nova amostra, 4 são do rótulo A e 3 do rótulo B. Portanto, como existem mais vizinhos do rótulo A, a nova amostra receberá o mesmo rótulo deles, ou seja, A.

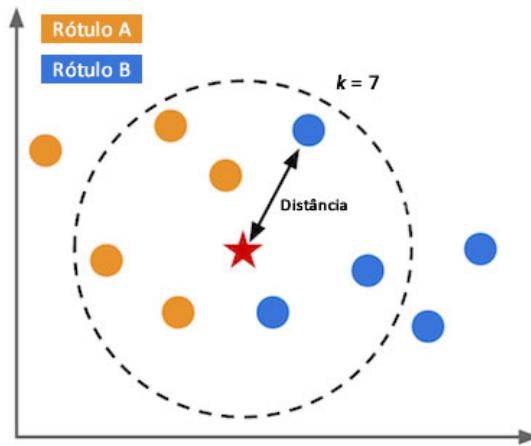


Figure 2.2: exemplo de classificação do KNN com dois rótulos de classe e  $k = 7$

Dois pontos chaves que devem ser determinados para aplicação do KNN são: a métrica de distância e o valor de  $k$ . Para métrica de distância a mais utilizada é a distância Euclidiana, descrita por:

$$D_E(p, q) = \sqrt{(p_1 - q_1)^2 + \cdots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^n (p_i - q_i)^2} \quad (2.1)$$

mas também podemos utilizar outros tipos de distância, como a distância de Minkowsky:

$$D_M(p, q) = \left( \sum_{i=1}^n |p_i - q_i|^r \right)^{\frac{1}{r}} \quad (2.2)$$

<sup>2</sup>A branch and bound algorithm for computing k-nearest neighbors. IEEE Transactions on Computers, v. 100, n. 7, p. 750–753, 1975

ou a distância de Chebyshev:

$$D_C(p, q) = \max_i \{|p_i - q_i|\} \quad (2.3)$$

Em ambas as equações, são dois vetores  $n$ -dimensionais.  $P = (p_1, \dots, p_n)$  e  $Q = (q_1, \dots, q_n)$  e  $r$  é uma constante.

No exemplo da Figura 2.2, essa distância seria calculada entre as bolinhas (azuis e laranjas) e a estrela (a nova entrada). Como o exemplo é 2D, cada ponto teria seu valor em  $x$  e em  $y$ . Para problemas com dimensões maiores a abordagem é exatamente a mesma.

Em relação ao valor  $k$ , não existe um valor único para a constante, a mesma varia de acordo com a base de dados. É recomendável sempre utilizar valores ímpares/primos, mas o valor ótimo varia de problema para problema. A maneira mais simples de encontrar um valor de  $k$  é simplesmente testar um conjunto de valores e encontrá-lo empiricamente. Um detalhe importante é que existe uma possibilidade, mesmo para  $k$  não par, de ocorrer empates. Quando isso ocorrer, neste trabalho você deve considerar a **primeira classe do empate**. Por exemplo, imagine que temos uma nova amostra na qual os vizinhos mais próximos para três rótulos são  $\{1, 4, 4\}$ . Neste caso, temos  $k = 9$  e os rótulos 2 e 3 estão empatados. Com isso, de acordo com nossa regra, você escolher o rótulo 2.

O pseudocódigo do KNN é descrito a seguir:

- 
- 1 inicialização:**
  - 2     Preparar conjunto de dados de entrada e saída
  - 3     Informar o valor de  $k$ ;
  - 4 para** cada nova amostra **faça**
  - 5     Calcular distância para todas as amostras
  - 6     Determinar o conjunto das  $k$ 's distâncias mais próximas
  - 7     O rótulo com mais representantes no conjunto dos  $k$ 's vizinhos será o escolhido
  - 9 fim para**
  - 10 retornar:** conjunto de rótulos de classificação
- 

## 2.3 Métricas de desempenho de classificadores

Quando desenvolvemos um classificador e aplicamos ele em um determinado problema, temos que verificar o desempenho do mesmo para o problema em questão. Neste trabalho vamos utilizar apenas duas métricas, a acurácia e a matriz de confusão.

### 2.3.1 Acurácia

Essa é a métrica mais simples, basta comparar quantas amostras certas existe no universo total. Por exemplo, imagine que temos 100 amostras para testar no nosso classificador. Nossa algoritmo acerta

81 e erra 19. Portanto, a acurácia do classificador para esse conjunto de teste é de 81% ( $\frac{81}{100}$ ), ou seja, generalizando:

$$Acc = \frac{A}{T} \quad (2.4)$$

Sendo  $A$  o total de acertos e  $T$  o total de amostras submetidas.

### 2.3.2 Matriz de confusão

A matriz de confusão já é uma medida de avaliação um pouco mais sofisticada, mas nada muito complicado. A matriz de confusão nada mais é do que uma tabela que permite visualizar o desempenho do classificador. Cada linha da matriz representa o rótulo das amostras que o classificador previu e cada coluna representa o rótulo real. Para facilitar a compreensão, vamos a um exemplo. Imagine que, de acordo com um conjunto de dados, vamos aplicar um classificador para prever se uma pessoa possui câncer ou não. Uma matriz de confusão para este problema pode ser a seguinte:

Preditiva/Correta	Sim	Não	Total
Sim	60	25	85
Não	11	190	201
Total	71	215	286

A pergunta a ser respondida pelo classificador é: a amostra (pessoa) possui câncer? Logo, o mesmo classifica em **Sim** ou **Não**. Para não criar uma confusão com a matriz de confusão, olhe para a diagonal principal. Das 85 amostras nas quais a amostra possui câncer, nosso classificador fictício rotulou corretamente 60 delas. Similarmente, das 201 nas quais a amostra não possui câncer, ele classificou corretamente 190. Essa informação encontra-se na diagonal principal. Perceba que a soma da linha e coluna entitulada **Total** obrigatoriamente deve ser igual ao número de amostras; no caso, 286. Por fim, é possível perceber que, dos 36 erros do classificador, 25 atingem o caso em que a pessoa tem câncer e o classificador diz que não. É para isso que serve a matriz de confusão, para detectar esses casos. Não vamos entrar em detalhes técnicos mas, a título de curiosidade, esses 25 erros são mais graves do que os 11 da linha de cima. Por que? Bem, prever que uma pessoa **não** tem câncer, mas na verdade ela tem é muito pior do que o contrário. Imagine que seu programa avise uma pessoa que ela não tem a doença. Ela vai pra casa e não se submete ao tratamento, que deveria ser urgente. Com a matriz de confusão é possível calcular os falsos positivos, verdadeiros negativos etc. Caso tenha interesse no tema, veja os links de apoio que estão no final desta documentação.

## 2.4 Detalhes da implementação

Sua tarefa neste trabalho será implementar o KNN, descrito anteriormente, utilizando a linguagem C. **Não é permitido** o uso de bibliotecas externas para computação de matrizes e afins. As bibliotecas necessárias para a implementação são as trabalhadas em sala de aula. Seu programa receberá um padrão de entrada e deverá retornar um padrão de saída que será descrito nas seções a seguir.

## 2.5 Padrão de entrada

Seu programa deverá ler três arquivos de entrada. O primeiro arquivo será o `config.txt` e estará na pasta raíz do seu código. A primeira linha do arquivo será o path para uma base de dados de

treinamento e a segunda linha o path para uma base de dados de teste. Ambos os arquivos deverão ser carregados pelo seu programa. As demais linhas, são parâmetros de configuração do KNN, ou seja:

- Valor de  $k$  (**int**)
- Tipo de distância do KNN, E para Euclideana, M para Minkowsky e C para Chebyshev (**char**)
- Se a distância for de Minkowsky, será informado também o valor de  $r$  (**float**)

Os valores serão separados por vírgula. Um exemplo do arquivo pode ser observado na sequência:

```
config.txt
-----
<path_base_de_dados_treino>.csv
<path_base_de_dados_teste>.csv
<path_arquivo_saida>
3, M, 2
4, D
5, C
8, D
-----
```

No caso, cada linha de configuração deverá ser uma execução do seu algoritmo para a base de dados informada nas duas primeiras linhas. Neste arquivo de exemplo, seu algoritmo deve ser executado 5 vezes, cada execução respeitando os parâmetros informados.

Os outros três arquivos de entrada serão a base de dados de treino e test com paths informados no arquivo de configuração e o caminho para você salvar o seu arquivo de saída do programa. Fazendo uma analogia com a Figura 2.2, a sua base de entrada de treinamento serão as bolinhas azuis e laranjas e a base de teste serão estrelas que você deseja rotular. Ambos os arquivos terão formato .csv e possuem o mesmo padrão exemplificado a seguir:

```
dados_entrada{treino,teste}.csv
-----
feature11, feature12, feature13, ..., feature1M, rotulo1
.
.
.
featureM1, featureM2, featureM3, ..., featureMN, rotuloN
-----
```

Perceba que o arquivo possui  $M \times N$  entradas e pode variar de acordo com a base de dados em questão. Cada linha representa uma amostra (uma bolinha ou estrela). O último valor da linha representa o rótulo da amostra e os demais são features da mesma (por exemplo, comprimento de sepálas e petálas dos exemplos da íris discutido na sessão de classificação de dados). Perceba também que problemas diferentes podem conter número de features diferente, mas obrigatoriamente terá apenas 1 rótulo.

As bases de treino e teste terão exatamente o mesmo formato. Em teoria, o teste não deveria possuir rótulo, uma vez que desejamos rotular ele. Todavia, para computar as métricas do classificador, é necessário a entrada dos rótulos. Portanto, o seu algoritmo vai classificar essas novas amostras do teste e quando terminar você deve comparar o rótulo que o classificador retornou com o rótulo real.

## 2.6 Padrão de saída

Seu algoritmo deve produzir um ou mais arquivos de saída chamado `resultados_X.txt`. O valor de X depende do número de execuções do seu algoritmo descrito no `config.txt`. Por exemplo, no arquivo de configuração da página 9, o seu código deve ser executado 4 vezes, uma para cada valor de k e suas determinadas distâncias. Sendo assim, no arquivo de resultados você deve reportar a acurácia (**utilize apenas 2 casas decimais**) do seu classificador e a matriz de confusão do mesmo. Na sequência, você deve imprimir a resposta do classificador para cada base amostra da base teste, uma por linha. Por exemplo, imagine que um arquivo e teste possua 5 amostras com 3 rotulos. Seu arquivo de predição deve retornar:

```
predicao_X.txt
```

```
-----  
0.95
```

```
1 0 0
```

```
1 2 0
```

```
0 0 1
```

```
rotulo1
```

```
rotulo2
```

```
rotulo3
```

```
rotulo4
```

```
rotulo5
```

A sequência das amostras no arquivo de resultado deve respeitar a mesma sequência do arquivo de teste, ou seja, o primeiro rótulo apresentado no arquivo de resultados deve ser referente a primeira amostra do arquivo de teste e assim por diante. **Lembre-se** de salvar seu arquivo de predição no <path\_arquivo\_saida> indicado no `config.txt`.

Respeite os padrões, inclusive as **quebras de linha**. Isso é muito importante no momento da correção do seu trabalho, que será automático. Caso você não siga o padrão, sua saída **não casará** com o padrão de testes e sua nota será afetada. Até dia 01/10 será disponibilizado uma bateria de teste para você validar o seu trabalho antes da submissão.

## 2.7 Regras gerais

O trabalho poderá ser **realizado em dupla**. Todavia, a nota é individual. Além da correção técnica de funcionamento e inspeção do código, será realizado uma entrevista com a dupla. Nessa entrevista serão realizadas algumas perguntas na quais definirão a nota de participação de cada um. Atente-se que se você comprou o trabalho ou teve um grande amigo muito bem intencionado que decidiu fazer o trabalho pra você, sua nota será corrigida na entrevista e por inspeção ao código enviado.

De maneira alguma tente copiar o trabalho de uma outra dupla. Todos os códigos serão submetidos a um **detector de plágio**. Se algum plágio for detectado, sua nota sera **zero**.

**Identifique o seu código!** Organização e modularização serão critérios de avaliação.

Qualquer situação extraordinária será levada em consideração no momento da correção.

### 2.7.1 Nota e critério de avaliação

A avaliação do trabalho será realizada em duas etapas:

- **Correção automática:** bateria de testes (algumas baterias de testes serão disponibilizadas no site da disciplina). Avaliação objetiva;
- **Entrevista:** análise e explicação do código por parte dos alunos. Avaliação subjetiva.

A nota final do trabalho será calculada da seguinte forma:

$$\text{NotaFinal} = \text{NotaCorreçãoAutomática} \times \text{NotaEntrevista} + \text{FatorDeAjuste} \quad (2.5)$$

A nota da correção automática será de 10 pontos e a da entrevista 1 ponto. O fator de ajuste é variável e existe apenas para corrigir possíveis injustiças na nota objetiva. Por exemplo, imagine que você esqueceu um espaço na saída do seu programa e a correção automática falhou. Neste caso é injusto não pontuar nesta bateria, logo, haverá um desconto, mas sua nota será ajustada. Lembre-se que o peso do trabalho na sua média final será de **30%**.

### 2.7.2 Submissão

A submissão deste trabalho será realizada através do [Github Classroom](#). Se você não sabe o que é o Github ou sabe mas nunca utilizou, você pode aprender através [deste tutorial](#). O Git é uma ferramenta utilizada por praticamente todas as empresas de desenvolvimento de software tornando-se **fundamental** para qualquer programador. Você deverá criar uma conta no GitHub para utilizar a ferramenta. Não se preocupe, é gratuito. Após criar a conta, e entender como o GitHub funciona, você deve ingressar no GitHub Classroom através [deste convite](#). Através deste link você deverá criar um time para dupla (atenção, times com nomes alusivos a discurso de ódio, mensagens políticas e mensagens nesta linha serão apagados. Seja um cara legal e evite isso neste espaço). Assim que o time for criado, um repositório será criado e você e sua dupla vão compartilhar esse repositório. Todas as alterações serão registradas dentro da Classroom. Após a data limite de submissão o último commit será salvo e não será possível enviar mais código para o repositório sem a permissão do administrador.

Para padronizar, o arquivo que possui a função main do seu código deve-se chamar `trab1.c`. Você também deve fornecer um arquivo `make` para determinar todas as regras de compilação do seu código. Esse arquivo é **fundamental** para que o executável possa ser gerado e os testes automáticos executados.

Seu trabalho poderá ser entregue até as **23:59 horas do dia 27/11/2018**. Não deixe para enviar seu trabalho nos momentos finais do prazo. É comum a ocorrência de problemas em virtude de erros na submissão. Logo, enviem com algumas horas (se possível dias) de antecedência para que haja tempo hábil para eventuais correções. Caso você não respeite a data limite, você perderá 2 pontos por dia de atraso, ou seja, se entregar no dia 28/11, sua nota máxima será 8 pontos, no dia 29, 6 pontos e assim por diante.

A correção do trabalho será realizada na medida que ele for entregue. Caso queira entregar ele antes e marcar a entrevista, isso será possível. O quanto antes você entregar o seu trabalho, mais rápido

ele será corrigido. Isso é muito bom para aqueles alunos que queiram adiantar o desenvolvimento do trabalho por conta de provas ou entregas de outras disciplinas. Vocês sabem que final de período é uma época muito complicada, então a sugestão é terminar o trabalho o quanto antes.

### **2.7.3 Dúvidas ou problemas encontrados na descrição do trabalho**

Qualquer dúvida ou problema encontrado na descrição do trabalho poderá ser sanada/apontada em sala de aula e/ou nos laboratórios em momentos oportunos, ou através da monitoria da disciplina. Mas por favor, leia mais de uma vez, converse com sua dupla antes de enviar um e-mail ou procurar o professor/monitor.

## **2.8 Fontes de referências**

Caso você se interesse pelo o assunto, existe um blog mantido pelo monitor da disciplina, chamado computação inteligente, em que discute vários conceitos de machine learning, otimização dentre outros. O endereço é <http://computacao-inteligente.com.br>.