FIAP GRADUAÇÃO

SISTEMAS DE INFORMAÇÃO

Computação Cognitiva

PROF. ANTONIO SELVATICI

Versão 3 – Fevereiro/2017

SHORT BIO

É engenheiro eletrônico formado pelo Instituto Tecnológico de Aeronáutica (ITA), com mestrado e doutorado pela Escola Politécnica (USP), e passagem pela Georgia Institute of Technology em Atlanta (EUA). Desde 2002, atua na indústria em projetos nas áreas de robótica, visão computacional e internet das coisas, aliando teoria e prática no desenvolvimento de soluções baseadas em Machine Learning, processamento paralelo e modelos probabilísticos. Desenvolveu projetos para Avibrás, IPT e Systax.

PROF. ANTONIO SELVATICI
profantonio.selvatici@fiap.com.br

2. MACHINE LEARNING

Ambiente para classificação de padrões

- As técnicas de reconhecimento de padrões que estudaremos serão executadas em um ambiente de comandos denominado Octave
- O programa Octave possui uma grande quantidade de ferramentas para a resolução de problemas de álgebra linear, encontra as raízes de equações não lineares, integra funções ordinárias, manipula polinômios, integra equações diferenciais ordinárias e equações diferenciais algébricas.
- Como linguagem de programação e ambiente de execução, assemelha-se ao Python, R, e, especialmente, ao Matlab, cuja sintaxe de script foi adotada pelo Octave.
- Permite ainda gerar e visualizar gráficos 2D e 3D a partir de dados
- GNU Octave é um software opensource, e pode ser estendido através das funções definidas pelo usuário escritas na própria linguagem do Octave, ou usando módulos escritos em C + +, C, Fortran ou outras linguagens.

O que é Octave?

- Há várias respostas....
- **Octave** é uma linguagem de programação:
  - interpretada, de alto nível,
  - orientada a objetos
  - extensível, através de adições de pacotes ou bibliotecas
  - permite chamadas a funções de C e Fortran de forma simples
- **Octave** é um ambiente interativo de comando
  - Executa uma grande variedade de métodos numéricos
  - Produz gráficos 2D e 3D
  - Considerado a versão opensource do Matlab

Hello World!

- Executar um comando no GNU Octave pela primeira vez
  - Digitar : `display("Hello World!")` -> veja a saída
- Dependendo da expressão, o sistema pode responder através de output no próprio console ou através de uma janela gráfica
- O ambiente interativo funciona como uma calculadora. Vamos executar algumas contas:
  - Ex: 5+3, 9/2, 4.24*3.13
- O Octave não mostra no console os resultados de comandos terminados em ponto-e-vírgula (;), daí o uso de `display()`

Expressões aritméticas e numéricas

- Operadores binários: +, -, *, /, ^ ou ** (exponenciação)
- Operadores lógicos: >, >=, <, <=, ==, !=,
- Funções matemáticas: abs, sqrt, log, exp, log10
- Funções trigonométricas: sin, cos, tan, asin, acos, atan
- Arredondamento: round, ceil, floor
- Quantidades: `Inf`, `-Inf`, `NaN` (not a number), `pi`, `exp(1)`, `1i` (unidade imaginária)
- Exemplos:
  - `mod(5, 4)`
  - `log(2)`
  - `cos(pi)`
  - `ceiling(3.2)`
  - `0/0`
  - `1/Inf`
  - `1 == 2`
  - `3 > -4`

Variáveis

- Variáveis em Octave são similares às variáveis das diferentes linguagens de programação interpretadas
  - Não-tipadas: o tipo da variável não precisa ser declarado e é definido na atribuição de valores
  - A variável é sempre uma referência para um objeto na memória (lembrando que tudo é objeto)
  - O operador de atribuição é "="
  - Variáveis são sensíveis à caixa e não podem iniciar por números nem conter caracteres especiais
- Executar:
  - `x`
  - `x = 5+3`
  - `y2 = x - 3`
  - `x + y2`

Tipos de dados

- **Numéricos**: dados numéricos, que, tecnicamente podem ser inteiros ou ponto flutuante (double), mas quase sempre correspondem ao último caso
- **Lógicos**: as constantes **true** e **false** correspondem aos números 1 e 0, respectivamente
- **Caracteres**: são as strings, definidas por texto entre ' ' ou " "
- **Structures:** são conjuntos de dados não homogêneos cujos campos são acessados através dos nomes das propriedades,
- **Objetos**: representam instâncias de classes do Octave
- **NA:** representam dados ausentes (missing data)

Conjuntos (estruturas) de dados

- Sendo Octave um ambiente voltado para processamento numérico, temos grande interesse em trabalhar com conjuntos de dados
- Os conjuntos de dados suportados pelo Octave/Matlab são:
  - Matrizes: valores do mesmo tipo organizados por linhas e colunas
  - Arrays: são generalizações das matrizes para várias dimensões
  - Cell arrays: coleção de dados de diversos tipos
  - Structure arrays: tipos especial de structure onde cada elemento é análogo a uma coluna de uma tabela de banco dados

Matrizes

- São o tipo mais básico de estrutura de dados
  - Vetores são simpesmente matrizes com uma linha ou coluna
- Os operadores '[' e ']' concatenam valores para formar uma matriz
  - [1, 3.4, -2.9, 3] %% Matriz linha
  - ["A"; "B"; "C"; "D"] %% Matriz coluna
- Para criar vetores com um padrão, podemos usar ':'(dois pontos) – cria sequências com incrementos ou decrementos fixos.
  - 2.3:5
  - 1:3:10
- Para plotar um vetor x: `plot(x)`
- Para plotar dois vetores de mesmo comprimento (eixo **x** e eixo **y**)
  - `plot(x, y)`
- Para ativar ou desativar a plotagem sobre a janela gráfica atual:
  - `hold on/off`

Matrizes>> Operações com vetores

- Para saber o tamanho de um vetor, usamos a função "length(x)"
- As operações e funções lógico-aritméticas são indistintamente aplicadas a vetores ou a quantidades escalares, de forma que estas podem ser encaradas como um "vetor unidimensional"
  - `log(2)`
  - `log([1,2,exp(1),4])`
  - `[1 2 3 4 5 6] > 3`
- Podemos aplicar as operações aritméticas de matrizes
  - `[2,3,4] + [3,2,1]`
  - `[1,2;3,4] * [1 3]' % O operador ' executa a matriz transposta`
  - `[2,2;2,2] .* [3 3;3 3] % O ponto . indica operação elemento a elemento`
- Para descobrir o tamanho da matriz:
  - `[linhas, colunas]=size(matriz);`

Vetores >> acessando seus elementos

- O Octave/Matlab é muito flexível no acesso a elementos de uma matriz.
- A contagem dos índices inicia em 1
- Através de parênteses ( ) podemos acessar um ou mais elementos
- Acesso a um único elemento:
  - `x(3,4) % linha 3, coluna 4`
- Acesso a um conjunto de valores através de um vetor de índices:
  - `x(1:3, 2:end) % A keyword 'end' se refere ao último índice`
  - `x([2,5,6])`
- Usando um vetor lógico para acessar os elementos
  - `x(x>4)`

Estruturas de controle de fluxo

- Para executar um código iterando sobre os elementos de um vetor:

```
for i = [ 2 4 8 16 32]
    <Código a ser executado>
endfor
```

- Para construir vetores de números em sequência, usar dois pontos:

```
– for i = 1:10 display(i) endfor
```

- Para executar um código com condição de parada:

```
while (i < 10)
    <Código a ser executado>
endwhile
```

Classificação de padrões com Octave

Etapas da classificação de padrões

- A classificação de padrões envolve a sequência de procedimentos envolvendo o conjunto de dados a ser classificado:
  - Pré-processamento de dados
  - Extração de atributos
  - Classificação
- Nas etapas de **pré-processamento** e **extração de atributos**, devemos tratar a mídia de origem dos dados, seja imagens, vídeos, textos, etc., para extrair os vetores de atributos usados no treinamento do classificador ou na classificação propriamente dita
- Porém, vamos aqui trabalhar com os atributos numéricos já extraídos dos dados brutos, uma vez que a disciplina (ainda) não trata de processamento de imagens ou de som

Trabalhando com ML no Octave

- O Octave possui alguns pacotes para execução dos algoritmos de Machine Learning, porém a facilidade em trabalhar com vetores e matrizes o torna atrativo para implementação dos mesmos
- Em primeiro lugar, é necessário importar os dados a serem empregados na classificação de padrões.
- Vamos usar os conjuntos de dados no formato CSV disponíveis no site http://archive.ics.uci.edu/ml/
  - Tomar cuidado para apagar linhas em branco no final do arquivo
- Para importar as colunas de atributos do formato CSV para o Octave, usamos a função `textread`
  - `[atr1, atr2,...,atrN] = textread(arquivo.csv, formato, propriedades...)`
    - O formato é uma string que usa um padrão similar ao do printf para indicar o formato de cada atributo, se é inteiro (%d), ponto flutuante (%f), string (%s), etc.
  - Exemplo: importando os atributos do dataset **Iris**
    - `[slen,swid,plen,pwid,classe] = textread('iris.data','%f%f%f%f%s','delimiter',',');`

Filtragem dos atributos

- Após a importação dos valores dos atributos, vamos analisá-los para identificar quais são os mais relevantes.
  - Quanto mais atributos usamos, mais precisa deverá ficar a classificação, porém maior o custo computacional e de memória usado pelos algoritmos, principalmente na fase de treinamento
  - Dependendo do tipo de classificador empregado, o algoritmo de treinamento pode demorar muito para finalizar, ou simplesmente não convergir para o resultado esperado caso o número de atributos seja muito grande.
  - Aos problemas que podem ser causados pelo excesso no número de atributos empregados chamamos de maldição da dimensionalidade
- O objetivo aqui é identificar se há um ou mais atributos que poderiam ser descartados sem que haja prejuízo da classificação
- Para tanto, vamos fazer uma análise visual dos dados

Análise visual dos dados

- Vamos usar a função `scatter` do Octave para visualizar os atributos dois a dois através de um gráfico de dispersão
  - Para verificar o uso da função, digitar `help(scatter)`
- Aqui usamos: `scatter(X, Y, S, C)`, onde
  - `X`: vetor com os dados do eixo **x**
  - `Y`: vetor com os dados do eixo ***y***
  - `S`: tamanho dos pontos do gráfico em pontos (default = 8)
  - `C`: string com a cor dos pontos do gráfico
    - `'K'` para preto, `'Y'` para amarelo, `'B'` para azul, e assim por diante
    - Para mais informações sobre essas opções, use a ajuda do comando `plot`

Análise visual do dataset iris

- Para cada par de atributos, vamos analisá-los plotando o gráfico de dispersão dos atributos de dois em dois , usando cores diferentes para classes diferentes
- Primeiramente, vamos encontrar os índices correspondentes a cada classe na forma de vetores lógicos
  - `idx_setosa = strcmp(classe,"Iris-setosa");`
  - `idx_virginica = strcmp(classe,"Iris-virginica");`
  - `idx_versicolor = strcmp(classe,"Iris-versicolor");`
- A função `strcmp` retorna 1 (ou `true`)quando os textos são idênticos, e 0 (ou `false`) quando são diferentes. A função `strcmpi` faz o mesmo, porém é insensível a letras maiúsculas ou minúsculas.
- Antes de iniciar a plotagem do gráfico, vamos usar a função `hold`.
  - `hold on`: os gráficos são plotados na mesma janela
  - `hold off`: os gráficos são plotados em uma nova janela

Plotagem do primeiro par de atributos

```
hold on %mantém a mesma janela gráfica
idx = idx_setosa; %usa apenas os índices com a classe 'setosa'
scatter(slen(idx),swid(idx),5,'K',"filled")
idx = idx_virginica; %usa apenas os índices com a classe 'virginica'
scatter(slen(idx),swid(idx),5,'R', "filled")
idx = idx_versicolor; %usa apenas os índices com a classe 'versicolor'
scatter(slen(idx),swid(idx),5,'G', "filled")
title("Sepal Length x Sepal Width") %acrescenta o título
legend("Setosa","Virginica","Versicolor") %acrescenta legenda
hold off %libera a janela gráfica
```

- Para plotar novos gráficos como esse, usar antes o comando `figure` para criar uma nova janela gráfica, caso contrário os novos gráficos poluirão a janela gráfica previamente construída
- Caso a análise de dois atributos se mostre pouco capazes de separar pelo menos duas classes, podemos descartar um deles
- Caso um par de atributos se mostre muito eficaz ao separar todas as classes, podemos tentar iniciar por esse par apenas, e depois acrescentar novos atributos se for necessário

Normalização de dados

- Observe que as tratar as distâncias em todas as direções de forma igualitária não é razoável, uma vez que pequenas diferenças em determinados atributos são importantes para definir a classe da amostra, porém diferenças maiores aplicadas a outros atributos parecem não fazer tanta diferença
- Dessa forma, melhores resultados serão alcançados se fizermos um ajuste de escala nos dados, de forma equalizar a importância dos atributos para efeito de classificação. A esse ajuste vamos chamar de **normalização** dos dados.
- Uma normalização simples, mas efetiva, é aquela que torna cada vetor de atributos um conjunto de dados de média zero e desvio padrão unitário
  - Para tanto, basta deduzir os valores de cada vetor de atributos de sua média (**`mean`**) e dividir pelo desvio padrão (standard deviation, ou **`std`**)
  - No Octave: `vetorNorm=(vetor - mean(vetor))/std(vetor);`
- A mesma normalização deve ser aplicada tanto às amostras de treinamento quanto nas amostras de teste

Média e desvio padrão

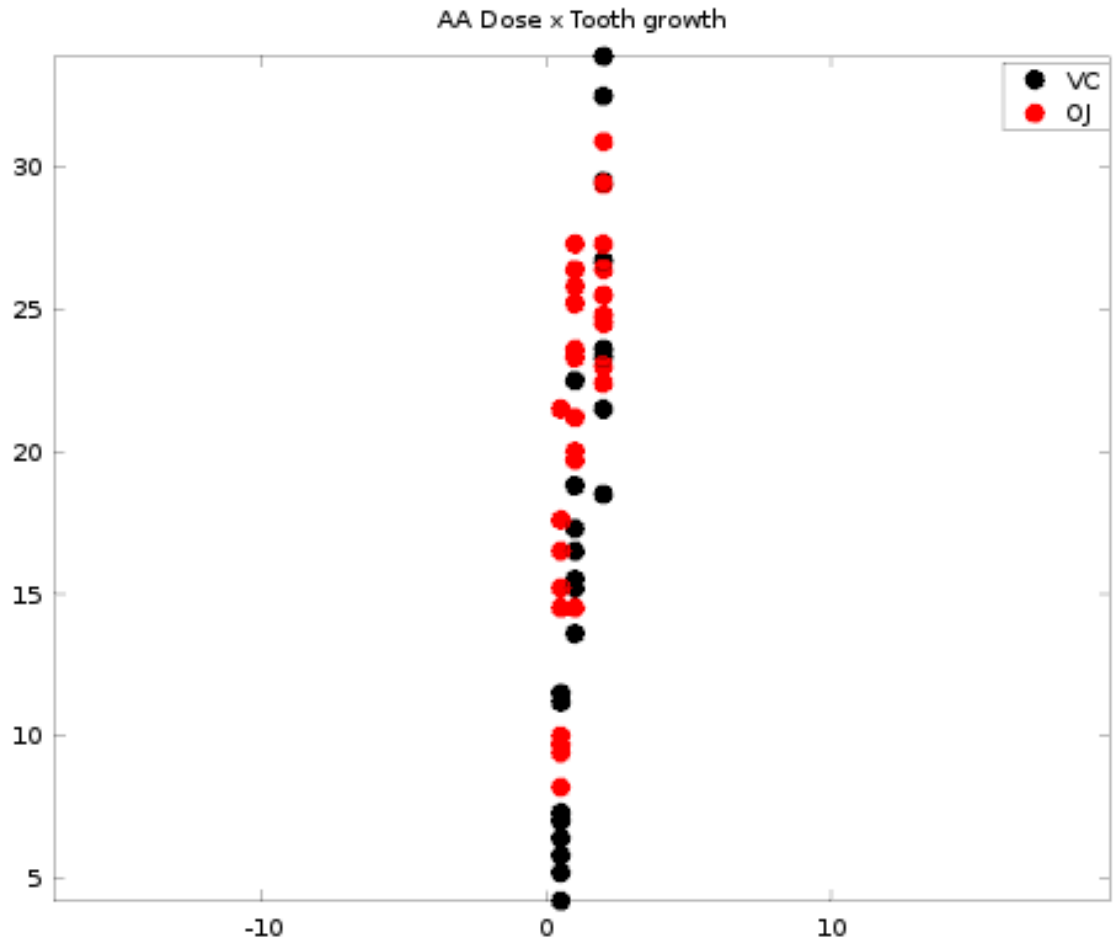
- Dado um vetor de dados $\mathbf{x} = (x_1, x_2, x_3, \ldots, x_n)$, podemos calcular:
  - Sua média, dada por

$$\bar{x} = \sum_{i=1}^{n} x_i = \frac{x_1 + x_2 + x_3 + \cdots + x_n}{n}$$

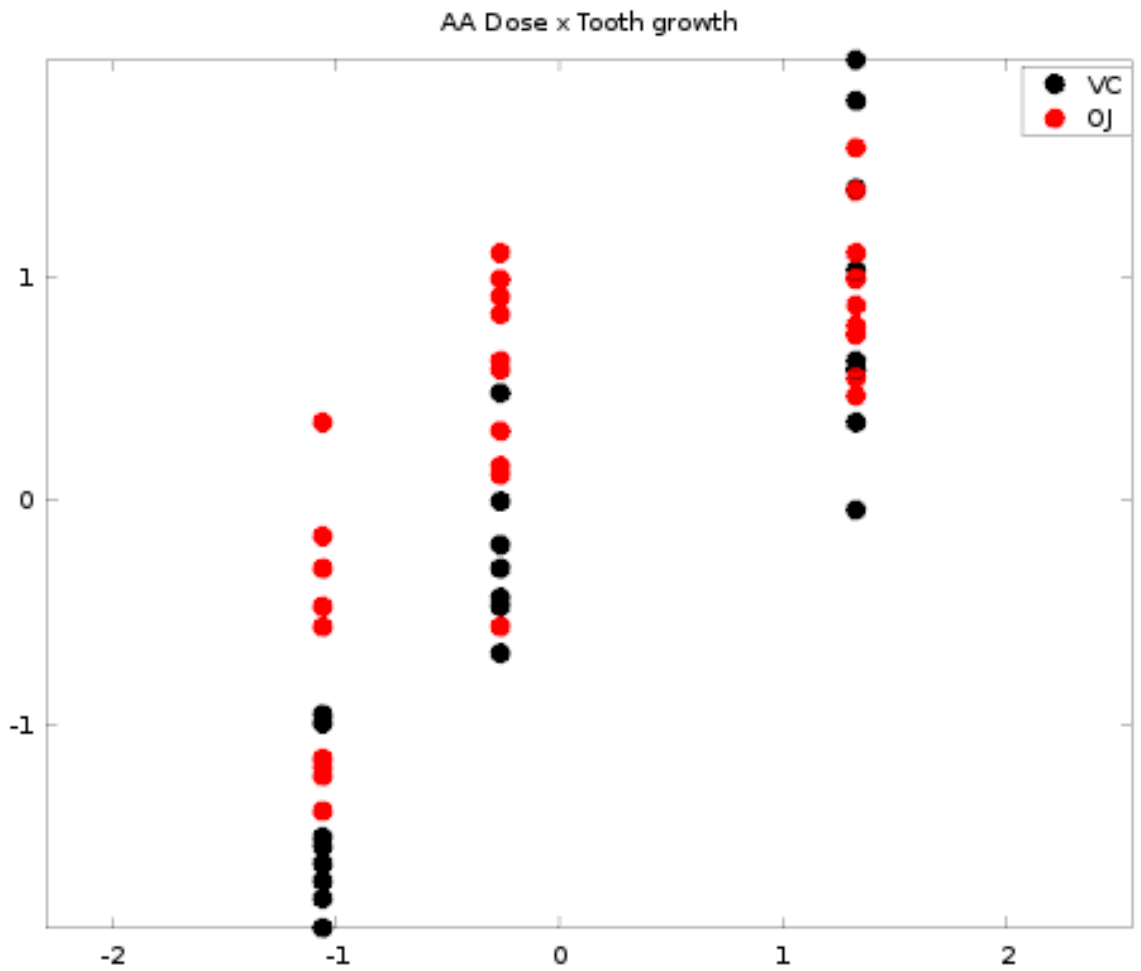
  - Seu desvio padrão, dado por

$$\sigma_x = \sqrt{\sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n-1}} = \sqrt{\frac{x_1^2 + x_2^2 + \cdots + x_n^2 - n\bar{x}^2}{n-1}}$$

Exemplo de espaço de atributos antes da normalização, com escalas iguais nos dois eixos



Exemplo de espaço de atributos após a normalização, com escalas iguais nos dois eixos



Classificação através de kNN

- A implementação da classificação por **k-vizinhos mais próximos** (ou **kNN**) usando o Octave é bastante simples
- Para uma nova amostra de teste **x**, devemos:
  1. Encontrar o vetor de distâncias das amostras de treinamento com relação a **x**;
  2. Ordenar o vetor de distâncias encontrado para determinar os vizinhos mais próximos; e
  3. Encontrar a classe que mais aparece na vizinhança determinada
- Para o passo 1, vamos aproveitar as facilidades de cálculo matricial fornecidas pelo Octave, representando tanto a amostra de teste **x** quanto as amostras rotuladas por matrizes
- A matriz M das amostras de treinamento é dada pela concatenação dos vetores de atributo como colunas dessa matriz:
  - No caso de usarmos os atributos "petal lenght" e "petal width" já normalizados:
  - `M = [plen pwid];`

Calculando as distâncias entre dois vetores de mesmas dimensões

- Para calcular a distância euclidiana entre dois vetores n-dimensionais $\mathbf{a} = (a_1, a_2, a_3, \ldots, a_n)$ e $\mathbf{b} = (b_1, b_2, b_3, \ldots, b_n)$, fazemos a raiz quadrada da soma das diferenças:

$$\sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_n - b_n)^2}$$

Implementação do kNN no Octave

- Cada linha da matriz M de corresponde a uma amostra de treinamento.
- Para calcularmos as diferenças , temos que transformar a amostra de teste (vetor) em uma matriz com as mesmas dimensões.
  - A amostra de teste deve ser transformada em uma matriz-linha e depois normalizada com os parâmetros do treinamento, por exemplo:
    `x = [(4.7-mean(plen))/std(plen), (1.9-mean(pwid))/std(pwid)];`
  - Transformação em matriz com nr linhas, onde nr é o número de amostras de treinamento:
    `X = repmat(x, rows(M),1);`
  - Matriz de diferenças: `D = M - X;`
- Cálculo do vetor de distâncias ao quadrado, onde a soma é feita ao longo da segunda dimensão (colunas) de D: `dists = sum(D.^2, 2);`
- Ordenação do vetor de distâncias e recuperação dos índices: `[Ord,I]=sort(dists);`
- Agora o vetor `I` possui os índices originais dos dados ordenados em `Ord`.
  - `Ord == dists(I);`
- Contando os `k` vizinhos pertencentes a uma certa classe `c`: `n=sum(classe(I(1:k))==c)`
- Se a classe for do tipo `string`, então a contagem de ocorrências para `k` vizinhos fica: `n=sum(strcmp({classe{I(1:k)}},c))`

FIAP