Questão 5

Mateus Pontes Mota

11 de maio de 2018

Sumário

Capítulo 1

Enunciado

Pede-se a implementação da abordagem um-contra-todos com SVMs para classificar a flor Íris a partir da base de dados de Fisher, este classificador deve ter 4 entradas e uma saída. Utilizar 80% de dados para treinamento e 20% para teste.

Apresentar:

1. Explicação resumida do funcionamento do código para SVM (Incluso SMO)
2. Explicação da Implementação um-contra-todos.
3. Listagem comentada do arquivo .m.
4. Percentuais de acerto no conjunto de treinamento e de teste.

Capítulo 2

Descrição da Solução

Para solucionar este problema implementando em *MatLab* iniciamos carregando a base de dados e definindo os parâmetros, além disso separamos a base de dados em treinamento e teste de maneira aleatória utilizando a função *dividerand()*.

Listing 2.1: Carregamento dos dados e declaração dos Parâmetros

```
%% Clears
clc;
clear all;
load('FisherIris.mat');
%% Parametros
mtData = [FisherIris.SepalLength FisherIris.SepalWidth FisherIris.
    PetalLength FisherIris.PetalWidth double(FisherIris.Species)]; %conjunto
    de amostras
cK = 'l' ; %Tipo de Kernel
dC= 0.4; %C
dMaxIter=10; %max de iterações sem mudar o alpha
dTol=10^-5;
[nSamples,nCols]=size(mtData); %numero de amostras total e numero de colunas
     da matriz
[TrainInd,~,TestInd]=dividerand(nSamples,0.8,0,0.2);
mtTrain = mtData(TrainInd,:); %matriz de treinamento
mtTest = mtData(TestInd,:); %matriz de teste
mtX=mtTrain(:,1:end-1); %dados de entrada
mtD=mtTrain(:,end); %dados de saida
```

É necessário então criar as 3 SVMs que serão utilizadas na abordagem de um-contra-todos, onde cada SVM representa uma das classes possíveis de modo que a saída de cada SVM será a resposta se a amostra pertence ou não a uma determinada classe. Precisamos então alterar a saída desejada que será utilizada em cada treinamento para que tenha como valor +1 caso seja a classe desejada e -1 caso pertença as outras duas classes. Após isto criamos as 3 SVMs passando as entradas, saídas desejadas e parâmetros, utilizamos kernel. Este processo foi implementado da seguinte forma:

Listing 2.2: Ajuste dos Pesos

```
%% Criando as 3 SVMs

mtY1=mtD; mtY2=mtD; mtY3=mtD;
mtY1(mtY1~=1)=-1; %Mudança dos dados para SVM1
mtY2(mtY2~=2)=-1; %Mudança dos dados para SVM2
mtY2(mtY2==2)=1;
mtY3(mtY3~=3)=-1; %Mudança dos dados para SVM3
mtY3(mtY3==3)=1;
```

```
[W1,b1]=createSVM(dC,dTol,dMaxIter,mtX,mtY1,cK); %cria a SVM1
[W2,b2]=createSVM(dC,dTol,dMaxIter,mtX,mtY2,cK); %cria a SVM2
[W3,b3]=createSVM(dC,dTol,dMaxIter,mtX,mtY3,cK); %cria a SVM3
```

O código para criar a SVM encontra-se no apendice e implementa o algoritmo apresentado em <http://cs229.stanford.edu/materials/smo.pdf>. Após criada a SVM fazemos o calculo da saída de cada SVM e pegamos o maior valor dentre todas as saídas para que seja a saída final do nosso sistema. Este processo foi implementado da seguinte forma:

Listing 2.3: Um Contra Todos e Resultados

```
%% Um-contra-todos e teste
%treinamento
mtY=[mtX*W1'+b1 , mtX*W2'+b2, mtX*W3'+b3];
[~,I]=max(mtY,[],2);
I(I~=mtD)=0;
fprintf('Porcentagem do treinamento: %6.4f',nnz(I)/length(I))
%teste
mtX=mtTest(:,1:end-1); %matriz de entrada para teste
mtDTest=mtTest(:,end); %vetor de saida desejado
mtY=[mtX*W1'+b1 , mtX*W2'+b2, mtX*W3'+b3];
[~,I]=max(mtY,[],2);
I(I~=mtDTest)=0;
pCT= nnz(I)/length(I);
fprintf('\nPorcentagem de teste: %6.4f \n',pCT)
```

Apêndice A

Códigos

Listing A.1: Funções de Kernel

```
function ker=kernel(X,xj,k)
%Computa o kernel de X e xj de acordo com o parametro k que identifica a
%função de kernel utilizada
N=size(X,1);
ker=zeros(N,1);
if k=='g'
    for i=1:N
        ker(i,1)=exp(-norm(X(i,:)-xj)); %gaussiana
    end
elseif k=='l'
    for i=1:N
        ker(i,1)=X(i,:)*xj'; %linear
    end
elseif k=='p'
    for i=1:N
        ker(i,1)=(X(i,:)*xj').^4; %polinomial de 4 ordem
    end
end

end
```

Listing A.2: Função para criar a SVM utilizando o SMO

```
function [ W,b ] = createSVM(C,tol,max_iter,X,Y,k  )
%%Esta função cria uma SVM baseada no algoritmo SMO
%C: parametro de regularização
%tol: tolerancia
%max_iter: max # de vezes que iteramos sobre alpha sem mudar
%X,Y : Entradas e  s a das  do conj de treinamento.
%k : funcao de kernel
N=length(Y);
alpha=zeros(N,1);
b=0;
counter=0;

while counter<max_iter
    N=size(Y,1);
    changedAlphas=0;
    for ii=1:N
        Ei=sum(alpha.*Y.*kernel(X,X(ii,:),k)) - Y(ii); %erro da amostra i:
            Ei = f(xi)-yi
```

```
18          %onde f(xi)=sum(alpha.y.k(x))
19          %eq. 2
20          if ((Ei*Y(ii) < -tol) && alpha(ii) < C) || (Ei*Y(ii) > tol && (alpha
                (ii) > 0))
21              j=ii;
22              %olha somente os ii diferentes de j
23              while(j==ii)
24                  j=randi(N);
25              end
26              Ej = sum(alpha.*Y.*kernel(X,X(j,:),k)) - Y(j); %% computa erro
                    da amostra j, eq. 2
27              alphaiOld = alpha(ii); %guarda o alpha indice i antigo
28              alphajOld = alpha(j); %guarda o alpha indice j antigo
29
30              %Encontra as margens L e H de modo que L <= alpha(j) <= H, para
31              %satisfazer a restrição 0 <= alpha(j) <= C :
32              %eq. 10 ou eq. 11
33              if Y(ii)~=Y(j)
34                  L = max(0,alpha(j)-alpha(ii));
35                  H = min(C,C+alpha(j)-alpha(ii));
36              else
37                  L = max(0,alpha(ii)+alpha(j)-C);
38                  H = min(C,alpha(ii)+alpha(j));
39              end
40              %continua para o prox i, pois n podemos balancear ai aj
41              if (L==H)
42                  continue
43              end
44              %Calcular Eta= 2*K(xi,xj) - K(xi,xi) - K(xj,xj)
45              %eq. 14
46              eta = 2*kernel(X(j,:),X(ii,:),k) - kernel(X(ii,:),X(ii,:),k) -
                    kernel(X(j,:),X(j,:),k);
47              %continua para o prox i caso nao seja um maximo
48              if eta>=0
49                  continue
50              end
51              %Calcula aj atraves da Eq. 12
52              alpha(j) = alpha(j) - (Y(j)*(Ei-Ej))/eta;
53              %Clip aj de modo que L <= aj <= H
54              if alpha(j) > H
55                  alpha(j) = H;
56              elseif alpha(j) < L
57                  alpha(j) = L;
58              end
59              %verifica se aj mudou consideravelmente
60              %caso contrario, nao precisamos ajustar ai
61              %e passamos para o prox i
62              if norm(alpha(j)-alphajOld) < tol
63                  continue
64              end
65              %Calcula ai pela Eq. 16
66              alpha(ii) = alpha(ii) + Y(ii)*Y(j)*(alphajOld-alpha(j));
67              %Calcula b1 e b2 usando eq. 17 e eq. 18
68              b1 = b - Ei - Y(ii)*(alpha(ii)-alphaiOld)*kernel(X(ii,:),X(ii,:)
                    ,k)...
69                  -Y(j)*(alpha(j)-alphajOld)*kernel(X(ii,:),X(j,:),k);
70              b2 = b - Ej - Y(ii)*(alpha(ii)-alphaiOld)*kernel(X(ii,:),X(j,:),
```

```
                k)...
71                  -Y(j)*(alpha(j)-alphajOld)*kernel(X(j,:),X(j,:),k);
72              %Calcula b de acordo com Eq. 19
73              if 0<alpha(ii)<C
74                  b=b1;
75              elseif 0<alpha(j)<C
76                  b=b2;
77              else
78                  b=(b1+b2)/2;
79              end
80              changedAlphas=changedAlphas+1;
81          end
82      end
83      if(changedAlphas==0)
84          break
85      end
86      counter=counter+1;
87      %%Utiliza somente as amostras uteis
88      X=X((find(alpha~=0)),:);
89      Y=Y((find(alpha~=0)),:);
90      alpha=alpha((find(alpha~=0)),:);
91  end
92  % Calculo do W
93  totalSum = 0;
94  N=length(alpha);
95  for jj=1:N
96      totalSum = totalSum + alpha(jj)*Y(jj)*X(jj,:);
97  end
98
99  W = totalSum;
100 b = Y(1) - X(1,:)*W';
101 end
102 \end
103
104 \begin{lstlisting}[language=Matlab, caption= Script completo]
105 %% Clears
106 clc;
107 clear all;
108 load('FisherIris.mat');
109 %% Parametros
110 mtData = [FisherIris.SepalLength FisherIris.SepalWidth FisherIris.
        PetalLength FisherIris.PetalWidth double(FisherIris.Species)]; %conjunto
        de amostras
111 cK = 'l' ; %Tipo de Kernel
112 dC= 0.4; %C
113 dMaxIter=10; %max de iterações sem mudar o alpha
114 dTol=10^-5;
115 [nSamples,nCols]=size(mtData); %numero de amostras total e numero de colunas
        da matriz
116 [TrainInd,~,TestInd]=dividerand(nSamples,0.8,0,0.2);
117 mtTrain = mtData(TrainInd,:); %matriz de treinamento
118 mtTest = mtData(TestInd,:); %matriz de teste
119 mtX=mtTrain(:,1:end-1); %dados de entrada
120 mtD=mtTrain(:,end); %dados de saida
121 %% Criando as 3 SVMs
122
123 mtY1=mtD; mtY2=mtD; mtY3=mtD;
```

```
124 mtY1(mtY1~=1)=-1; %Mudança dos dados para SVM1
125 mtY2(mtY2~=2)=-1; %Mudança dos dados para SVM2
126 mtY2(mtY2==2)=1;
127 mtY3(mtY3~=3)=-1; %Mudança dos dados para SVM3
128 mtY3(mtY3==3)=1;
129 [W1,b1]=createSVM(dC,dTol,dMaxIter,mtX,mtY1,cK); %cria a SVM1
130 [W2,b2]=createSVM(dC,dTol,dMaxIter,mtX,mtY2,cK); %cria a SVM2
131 [W3,b3]=createSVM(dC,dTol,dMaxIter,mtX,mtY3,cK); %cria a SVM3
132
133 %% Um-contra-todos e teste
134 %treinamento
135 mtY=[mtX*W1'+b1 , mtX*W2'+b2, mtX*W3'+b3];
136 [~,I]=max(mtY,[],2);
137 I(I~=mtD)=0;
138 fprintf('Porcentagem do treinamento: %6.4f',nnz(I)/length(I))
139 %teste
140 mtX=mtTest(:,1:end-1); %matriz de entrada para teste
141 mtDTest=mtTest(:,end); %vetor de saida desejado
142 mtY=[mtX*W1'+b1 , mtX*W2'+b2, mtX*W3'+b3];
143 [~,I]=max(mtY,[],2);
144 I(I~=mtDTest)=0;
145 pCT= nnz(I)/length(I);
146 fprintf('\nPorcentagem de teste: %6.4f \n',pCT)
```