[PSI5790, aula 4 parte 2. Início.]

Introdução ao Aprendizado de Máquina

Uma possível definição de aprendizado de máquina é:

> Aprendizado de máquina é o estudo de algoritmos que executam uma certa tarefa baseando-se em exemplos, sem usar instruções explícitas. (definição de [Wikipedia] simplificada)

Uma outra definição é:

> "Diz-se que um programa computacional aprende com a experiência *E* no que diz respeito a uma tarefa *T* e medida de desempenho *P*, se o seu desempenho na tarefa *T*, medida por *P*, aumenta com a experiência *E*." [Mitchell1997]

O aprendizado de máquina divide-se em duas categorias principais:

- Aprendizado supervisionado
- Aprendizado não-supervisionado

Também existem técnicas que não se enquadram em nenhuma dessas duas categorias, como o aprendizado auto-supervisionado. Neste curso, não estudaremos o aprendizado não-supervisionado.

1. Aprendizado supervisionado:

No aprendizado supervisionado, um “professor” ou “oráculo” fornece amostras ou exemplos de treinamento de entrada/saída (*ax*, *ay*). O “aprendiz” ou “algoritmo de aprendizado” *M* classifica uma nova instância de entrada *qx* baseado nos exemplos fornecidos pelo professor, gerando a classificação *qp*. Para cada instância de entrada *qx* existe uma classificação correta ou verdadeira *qy*, desconhecida pelo aprendiz. Se a classificação do aprendiz *qp=M*(*qx*) for igual à classificação verdadeira *qy* então o aprendiz acertou a classificação.

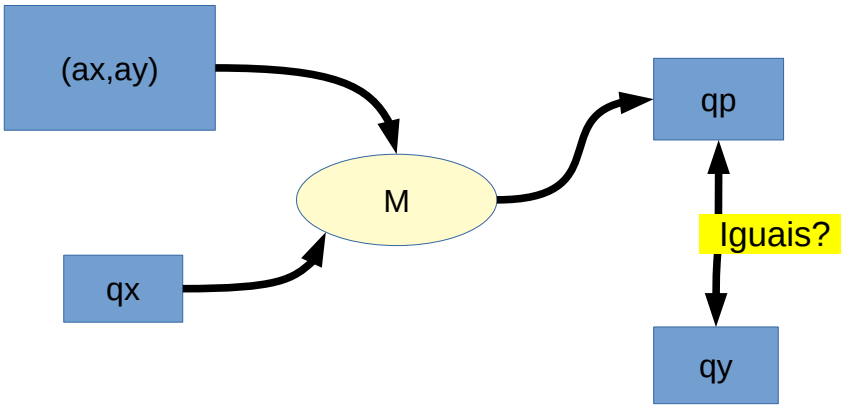


Figura: Esquema de aprendizado supervisionado.

O método de aprendizado supervisionado mais famoso é sem dúvida rede neural artificial. Porém, existem muitos outros métodos clássicos de aprendizado supervisionado que foram e continuam sendo usados. O estudo deles pode esclarecer melhor as técnicas modernas de inteligência artificial. Nas próximas aulas, veremos alguns métodos clássicos juntamente com suas aplicações.

OpenCV possui várias rotinas clássicas de aprendizado implementadas que podem ser chamadas de C/C++ ou de Python. Porém, aparentemente, as rotinas da biblioteca SkLearn de Python são mais confiáveis e flexíveis. Vamos usar as funções do OpenCV mas, quando toda vez que for necessário, usaremos aquelas do SkLearn.

Problema ABC

Para exemplificar o aprendizado de máquina, vamos considerar um problema muito simples: “Adulto”, “Bebê” ou “Criança” (ABC).

O aprendiz (ou o algoritmo de aprendizado de máquina) *M* é alimentado com a tabela 1, com conjunto de exemplos de entrada-saída ou (*ax*, *ay*):

{ (4,B), (15,C), (65,A), (5,B), (18,C), e (70,A) }.

Este conjunto indica o peso em kg dos indivíduos e a sua classificação em “Adulto”, “Bebê” ou “Criança”, mas o aprendiz *M* não sabe o que significam estes dados. Por exemplo (4,B) indica que há um indivíduo de 4 kg e sua classificação é “Bebê”. Depois, pede-se ao aprendiz *M* que classifique dado de teste *qx*=16.

O aprendiz pode adotar diferentes técnicas para classificar o indivíduo com atributo *qx*=16. Porém, uma técnica “razoável” seria procurar na tabela aquela instância mais parecida com 16. Fazendo isto, *M* encontra *ax*=15 cuja classificação é “C”. Então, *M* atribui rótulo “C” a 16, isto é, *M*(16) = “C”, pois é razoável atribuir rótulos iguais a instâncias semelhantes. Este método de aprendizado chama-se classificador vizinho mais próximo.

Este método é denotado como *k*-NN, onde *k* indica quantos vizinhos mais próximos serão procurados. O método descrito acima usa *k*=1, pois busca o vizinho mais próximo. Se *k* fosse 3, procuraria 3 vizinhos mais próximos da instância de teste e calcularia a maioria dos votos (moda) desses 3 vizinhos.

Tabela 1: Amostra de treinamento (*ax*, *ay*):

ax (features, atributos, características, entradas)	*ay* (labels, rótulos, saídas, classificações)
4	B
15	C
65	A
5	B
18	C
70	A

Tabela 2: Instâncias a classificar (*qx*), classificação verdadeira (*qy*) e classificação do aprendiz (*qp*).

Instância a processar ou a classificar (*qx*):	Classificação ideal ou verdadeira desconhecida (*qy*):	Classificação feita pelo aprendiz (*qp*):
16	C	C
50	A	A

Implementação do problema ABC

O programa abaixo resolve o problema ABC usando o classificador vizinho mais próximo implementado “manualmente” em C++, isto é, sem usar nenhuma função pronta de aprendizado das bibliotecas.

```
//abc2.cpp
#include "procimagem.h"
int main() {
  Mat_<float> ax = (Mat_<float>(6,1) <<  4, 15, 65, 5, 18, 70);
  Mat_<float> ay = (Mat_<float>(6,1) <<  1,  2,  0, 1,  2,  0);
  Mat_<float> qx = (Mat_<float>(3,1) << 16,  3, 75);
  Mat_<float> qy = (Mat_<float>(3,1) <<  2,  1,  0);
  Mat_<float> qp(3,1);

  for (int iq=0; iq<qx.rows; iq++) {
    float menorDist=FLT_MAX, menorAy;
    for (int ia=0; ia<ax.rows; ia++) {
      float dist=abs(ax(ia)-qx(iq));
      if (dist<menorDist) { menorDist=dist; menorAy=ay(ia); }
    }
    qp(iq)=menorAy;
  }
  cout << "qp:\n" << qp << endl;
  cout << "qy:\n" << qy << endl;
}
```

Executando o programa, obtemos:

```
qp: [2; 1; 0]  // Saidas geradas pelo programa
qy: [2; 1; 0]  // Rotulos verdadeiros
```

onde os rótulos 0, 1 e 2 representam respectivamente A, B e C. O programa conseguiu classificar corretamente os três indivíduos de pesos 16, 3 e 75 kg em C, B, A.

Exercício: Traduza o programa abc2.cpp para Python.

Exercício: Adapte o programa abc2.cpp para resolver o problema de classificação ABC quando é fornecida a altura do indivíduo em cm (em vez de peso).

Amostra de treinamento (*ax*, *ay*):

ax (features, características, entradas)	*ay* (labels, rótulos, saídas, classificações)
46	B
120	C
165	A
51	B
110	C
173	A

Instâncias a classificar (*qx*) e classificação verdadeira (*qy*).

Instância a processar ou a classificar (*qx*):	Classificação ideal ou verdadeira (*qy*):
60	B
168	A
105	C

Problema ABC com cor da pele

Considere novamente o problema de classificar um indivíduo como “Adulto”, “Bebê” ou “Criança”, mas desta vez são dados como entrada o peso em kg e a cor da pele (0 = escura, 100 = clara). Evidentemente, a cor da pele não tem nada a ver com a classificação A, B ou C, mas o algoritmo *M* não sabe disso.

ax		ay
peso	cor da pele	
4	90	B
15	50	C
65	70	A
5	7	B
18	3	C
70	80	A

qx		qy	qp (vizinho + px)
peso	cor da pele		
20	6	C	C
16	80	C	B
68	3	A	C
80	20	A	A
7	70	B	B
3	50	B	C

Nota para mim: Programas em ~/algpi/introaprend/pele

Repare na tabela acima que o algoritmo do vizinho mais próximo irá cometer 3 erros (marcados em amarelo). Na figura X, os pontos “o” são os dados de treino e os pontos “x” são os dados de teste. Os pontos “x” marcados em amarelo são dados de teste classificados incorretamente. O que se deve fazer para classificar corretamente todas as instâncias de teste?

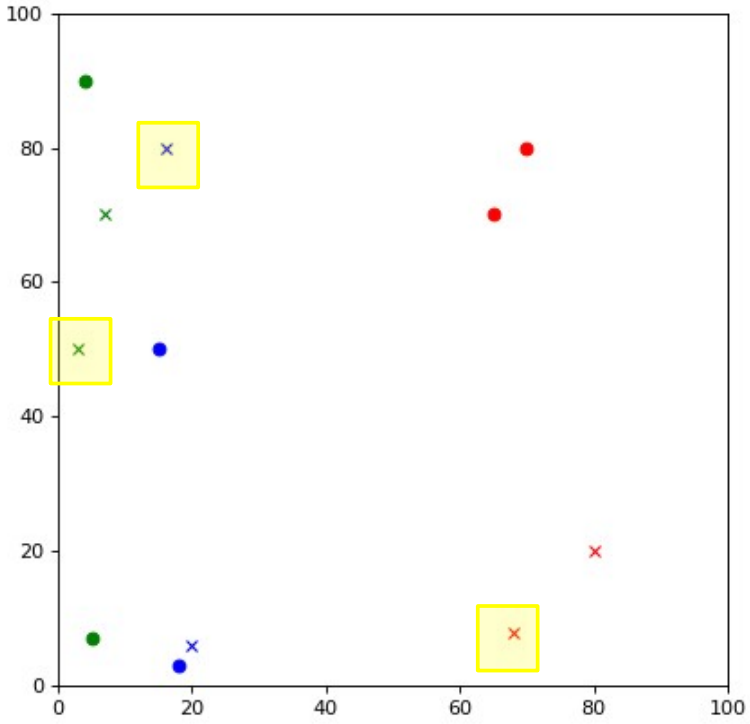


Figura X: Distribuição de pesos (eixo *x*) e cores da pele (eixo *y*). Pontos “o” indicam dados de treino. Pontos “x” indicam dados de teste. Vermelho é Adulto, verde é Bebê e azul é Criança. Os pontos marcados com retângulo amarelo indicam dados de teste que serão classificados incorretamente.

Exercício: Modifique o programa abc2.cpp acima para resolver o problema “ABC com cor de pele” e verifique que o programa realmente comete 3 erros.

Árvore de decisão

Seria possível que algum método de aprendizado de máquina consiga descobrir por si só que a “cor da pele” não influencia se o indivíduo é Adulto, Bebê ou Criança? A árvore de decisão consegue. Vamos resolver o problema “ABC com cor da pele” usando árvore de decisão.

A árvore de decisão faz sucessivos cortes no espaço dos atributos. Ela escolhe um atributo e um valor que mais ajudam a resolver o problema de classificação. Existem vários métodos diferentes para escolher o melhor atributo e valor de corte [Wikipedia]. Uma possibilidade é calcular “ganho de informação” (ou “redução de entropia”); outra possibilidade é usar índice Gini [aprendizagem-ead , anexo B].

Vou dar uma explicação intuitiva do ganho de informação. Considere o nosso problema da seção anterior com 6 amostras de treino: {2 Adultos, 2 Bebês e 2 Crianças}. Se, ao fazermos o primeiro corte, obtivermos dois subconjuntos com {1 Adulto, 1 Bebê e 1 Criança}, esse corte não ajudou em nada a resolver o problema de classificação. Porém, se o corte fizer resultar nos subconjuntos {2 Adultos, 0 Bebê e 0 Criança} e {0 Adulto, 2 Bebês e 2 Crianças}, ele separou Adultos das demais categorias, ajudando a resolver o problema (houve ganho de informação ou redução de desordem). A árvore de decisão procura escolher cortes que ajudem a resolver o problema.

A implementação de decision-tree em OpenCV2 só funciona se tiver mais de 100 amostras de treinamento. Assim, vamos usar a implementação de Python/SkLearn que não tem esta limitação.

```
 1 #dt2-pele.py
 2 import numpy as np
 3 from sklearn import tree
 4 import matplotlib.pyplot as plt
 5
 6 ax = np.matrix("4 90; 15 50; 65 70; 5 7; 18 3; 70 80",dtype=np.float32)
 7 ay = np.matrix("1;  2;  0; 1;  2;  0",dtype=np.float32)
 8 qx = np.matrix("20 6;  16 80; 68 8; 80 20; 7 70; 3 50",dtype=np.float32)
 9 qy = np.matrix(" 2; 2; 0; 0; 1; 1",dtype=np.float32)
10
11 arvore= tree.DecisionTreeClassifier()
12 arvore= arvore.fit(ax, ay)
13 qp=arvore.predict(qx)
14 print("qp: ", qp)
15 print("qy: ", np.ravel(qy))
16
17 fig=plt.figure(figsize=(8,6))
18 tree.plot_tree(arvore,filled=True,fontsize=10)
19 plt.tight_layout()
20 plt.show()
21 fig.savefig("dt2-pele.png")
```

/home/hae/algpi/introaprend/pele/dt2-pele.py

Este programa resolveu perfeitamente o problema “ABC com cor de pele”:

```
qp:  [2. 2. 0. 0. 1. 1.]
qy:  [2. 2. 0. 0. 1. 1.]
```

O programa Sklearn pode mostrar a árvore gerada:

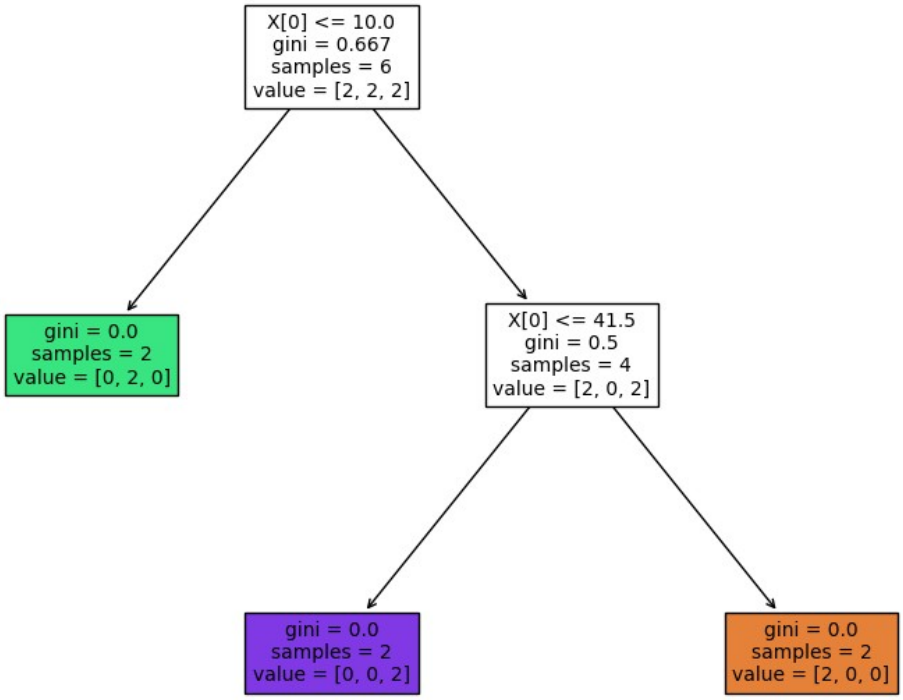


Figura: A árvore de decisão gerada pelo programa usando SkLearn/Python.

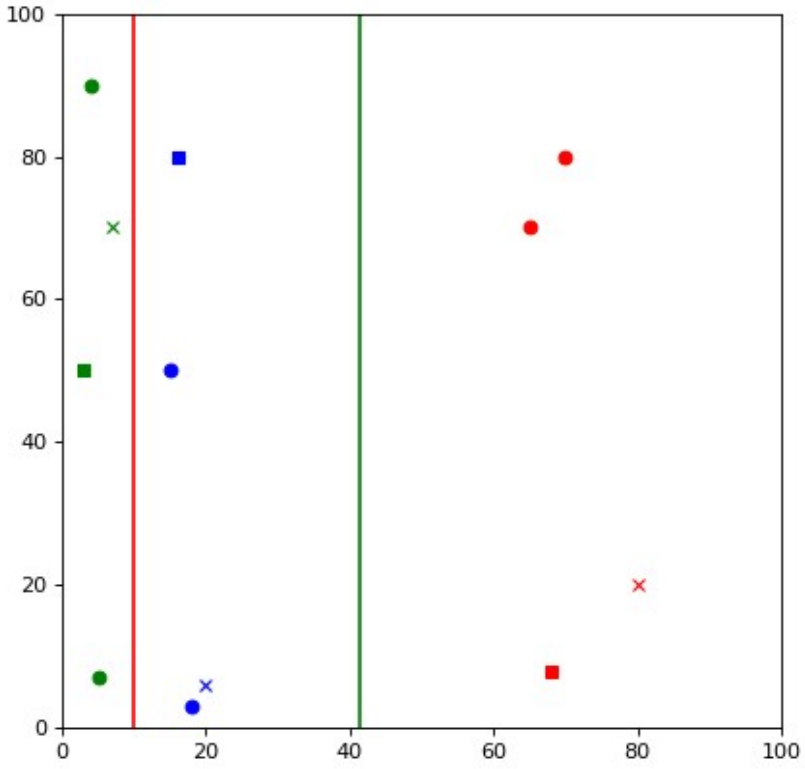


Figura: Superfície de decisão da árvore de decisão, com pesos (x) e cores da pele (y). Pontos “o” indicam dados de treino. Pontos “x” indicam dados de teste classificados corretamente por vizinho mais próximo. Pontos “■” indicam dados de teste classificados incorretamente por vizinho mais próximo. Vermelho é Adulto, verde é Bebê e azul é Criança. As linhas vermelho e verde indicam as duas linhas de corte.

Observando as figuras acima, descobrimos que os dois cortes foram feitos no eixo *x* (peso) e os valores das cortes são 10 (linha vermelha que separa Bebê de Crianças/Adultos) e 41.5 (linha verde que separa Crianças de Adultos). Nenhum corte foi feita no eixo *y* (cor de pele) pois este atributo é inútil para a classificação.

Dada uma instância a classificar, por exemplo qx=[20, 6], a árvore de decisão verifica se o primeiro atributo (X[0]=20) é <=10. Como é 20>10, navega para a sub-árvore da direita. Agora, verifica se o primeiro atributo (X[0]=20) é <=41.5. Como 20<41.5, navega para o nó da esquerda (em roxo). Nesse nó, estão armazenadas 2 exemplos de treino da classe “2” (value = [0, 0, 2]). Portanto, qx=[20, 6] é classificada pela árvore de decisão como classe “2”, isto é, M([20, 6])=“2” ou qp=“2”.

Exercício: Adapte o programa dt2-pele.py para resolver o problema de classificação ABC com cor da pele quando é fornecida a altura do indivíduo em cm (em vez de peso).

ax		ay
altura	cor da pele	
46	90	B
120	50	C
165	70	A
51	7	B
110	3	C
173	80	A

qx		qy
altura	cor da pele	
102	6	C
124	80	C
168	3	A
176	20	A
44	70	B
61	50	B

Leitura e gravação de matrizes float

Precisamos de funções para ler/escrever matrizes de float de/para arquivos texto em disco em C++/Python. Vamos adotar a seguinte convenção para representar uma matriz float como texto. Os dois primeiros números indicam os números de linha e coluna da matriz. Depois, aparecem os elementos da matriz separados por espaços ou finais de linhas. Por exemplo:

```
3 3
1 2 3
4 5 6
7 8 9
```

Figura F1: a.txt

A biblioteca *Cekeikon* possui as funções “le” e “imp” que respectivamente lê e imprime uma matriz float. Considere o seguinte programa:

```
1 //leimp.cpp
2 #include <cekeikon.h>
3 int main() {
4   Mat_<float> A = (Mat_<float>(3,3) <<  1,2,3,  4,5,6,  7,8,9);
5   imp(A,"a.txt");
6   Mat_<float> B;
7   le(B,"a.txt");
8 }
```

Programa P1.

Na linha 5, a função *imp* imprime matriz *A* em arquivo texto, obtendo o arquivo “a.txt” da figura F1. Na linha 7, a função *le* lê um arquivo texto numa matriz float. Precisamos de funções semelhantes em C++ e Python.

```
//procimagem.h 2024
(...)
void le(Mat_<float>& a, string nome) {
  int nl,nc,e;
  FILE *arq=fopen(nome.c_str(),"r");
  if (arq==NULL) { printf("Erro arquivo\n"); exit(1); }
  e=fscanf(arq,"%d%d",&nl,&nc);
  if (e!=2) { printf("Erro leitura\n"); exit(1); }
  a.create(nl,nc);
  for (int l=0; l<nl; l++)
    for (int c=0; c<nc; c++)
      e=fscanf(arq,"%f",&a(l,c));
      if (e!=1)  { printf("Erro leitura\n"); exit(1); }
  fclose(arq);
}
```

```
def le(nomearq):
  with open(nomearq,"r") as f:
    linhas=f.readlines()
  linha0=linhas[0].split()
  nl=int(linha0[0]); nc=int(linha0[1])
  a=np.empty((nl,nc),dtype=np.float32)
  for l in range(nl):
    linha=linhas[l+1].split()
    for c in range(nc):
      a[l,c]=np.float32(linha[c])
  return a
```

Programa P: Função que lê uma matriz float em C++ (a ser inserida em procimagem.h) e Python.

Exercício: A função *le* em Python não verifica erros de leitura. Acrescente a verificação.

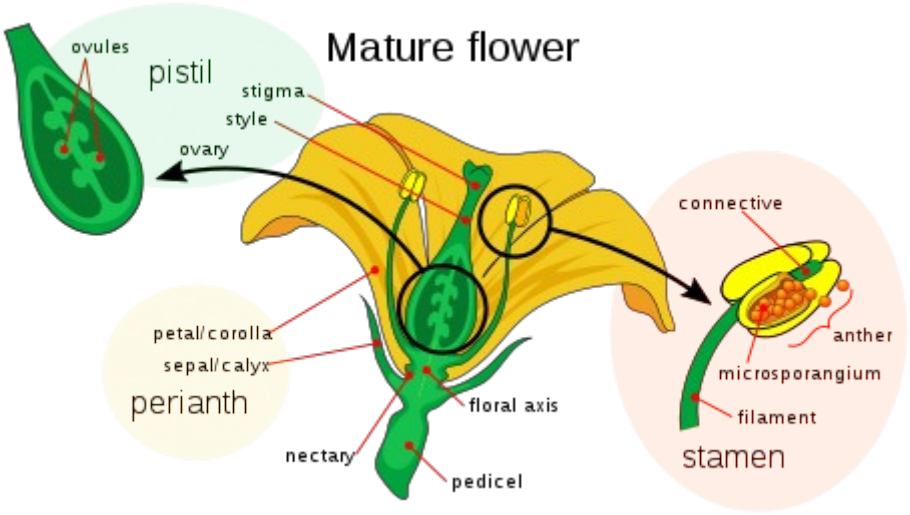
Classificação de flor “Iris”[1]

[Nota para mim: Este exemplo não é muito didático. Revisar.]

Para continuarmos o estudo dos algoritmos de aprendizado, vamos usar um problema um pouco mais complexo do que problema “ABC”. Vamos usar o conjunto de dados “Iris” que vem junto com o módulo SkLearn de Python.

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html
https://en.wikipedia.org/wiki/Iris_flower_data_set

Este conjunto de dados consiste em atributos extraídos de 50 amostras flores de cada uma das três espécies de “Iris”: Iris setosa, Iris virginica e Iris versicolor. Quatro atributos foram extraídos de cada amostra: comprimento e largura de sépalas e pétalas em centímetros. Esse conjunto de dados está disponível dentro do SkLearn.



De Wikipedia.

Abaixo, as fotos das três espécies de “Iris”:

Iris setosa (classe “0”)

Iris versicolor (classe “1”)

Iris virginica (classe “2”)

De Wikipedia

1 Programas em ~/algpi/introaprend/iris e ~/algpi/introaprend/iris2d

É difícil visualizar dados 4D para entender o funcionamento dos algoritmos de aprendizado. Assim, eliminei os dois primeiros atributos ficando somente com os dois últimos atributos: comprimento e largura da pétala. Além disso, separei os dados em treino (30 primeiros dados de cada espécie, *ax* e *ay*) e teste (20 últimos dados de cada espécie, *qx* e *qy*), obtendo 90 exemplos de treino e 60 exemplos de teste. Vamos chamar este novo problema de "Iris2D". Os quatro arquivos gerados são:

Linha a	ax.txt	ay.txt		Linha q	qx.txt	qy.txt
1	90 2	90 1		1	60 2	60 1
2	1.4 0.2	0		2	1.6 0.2	0
3	1.4 0.2	0		3	1.5 0.4	0
...	...	...		...	...	...
30	1.4 0.2	0		20	1.5 0.2	0
31	1.6 0.2	0		21	1.4 0.2	1
32	4.7 1.4	1		22	3.8 1.1	1
...	...	...		...	...	...
61	3.5 1	1		41	4.1 1.3	1
62	6 2.5	2		42	6.1 1.9	2
...	...	...		...	...	...
91	5.8 1.6	2		61	5.1 1.8	2

Os 6 arquivos assim obtidos (irisdata.txt, iristarget.txt, ax.txt, ay.txt, qx.txt, qy.txt) estão em:
http://www.lps.usp.br/hae/apostila/iris.zip
E podem ser baixadas em Linux com os comandos:
$ wget -U 'Firefox/50.0' http://www.lps.usp.br/hae/apostila/iris.zip
$ unzip iris.zip

Vizinho mais próximo

Podemos resolver o problema “Iris2D” usando o classificador vizinho mais próximo. O programa abaixo resolve o problema Iris2D usando o vizinho mais próximo escrito “manualmente” em C++, isto é, sem usar uma função pronta da biblioteca.

```
1  //nn_man.cpp
2  #include <cekeikon.h>
3  int main() {
4    Mat_<float> ax; le(ax,"ax.txt");
5    Mat_<float> ay; le(ay,"ay.txt");
6    Mat_<float> qx; le(qx,"qx.txt");
7    Mat_<float> qy; le(qy,"qy.txt");
8    Mat_<float> qp(qy.rows,qy.cols);
9    for (int iq=0; iq<qx.rows; iq++) {
10     float menorDist=FLT_MAX, menorAy;
11     for (int ia=0; ia<ax.rows; ia++) {
12       float dist=norm(ax.row(ia)-qx.row(iq));
13       if (dist<menorDist) { menorDist=dist; menorAy=ay(ia); }
14     }
15     qp(iq)=menorAy;
16   }
17   int erros=0;
18   for (int i=0; i<qp.rows; i++)
19     if (qp(i)!=qy(i)) erros++;
20   printf("Erros=%d/%d.   Pct=%1.3f%%\n",erros,qp.rows,100.0*erros/qp.rows);
21 }
```

```
1  //nn_man.cpp
2  #include "procimagem.h"
3  int main() {
4    Mat_<float> ax; le(ax,"ax.txt");
5    Mat_<float> ay; le(ay,"ay.txt");
6    Mat_<float> qx; le(qx,"qx.txt");
7    Mat_<float> qy; le(qy,"qy.txt");
8    Mat_<float> qp(qy.rows,qy.cols);
9    for (int iq=0; iq<qx.rows; iq++) {
10     float menorDist=FLT_MAX, menorAy;
11     for (int ia=0; ia<ax.rows; ia++) {
12       float dist=norm(ax.row(ia)-qx.row(iq));
13       if (dist<menorDist) { menorDist=dist; menorAy=ay(ia); }
14     }
15     qp(iq)=menorAy;
16   }
17   int erros=0;
18   for (int i=0; i<qp.rows; i++)
19     if (qp(i)!=qy(i)) erros++;
20   printf("Erros=%d/%d.   Pct=%1.3f%%\n",erros,qp.rows,100.0*erros/qp.rows);
21 }
```

/home/hae/algpi/introaprend/iris2d/nn_man.cpp

A linha 8 pede para criar a matriz *qp* e alocar para ele o mesmo número de linhas e colunas da matriz *qy*.

Você pode compilar este programa e linkar com Cekeikon e OpenCV2 com o comando:

```
Windows ou Linux> compila nn_man -c
```

Também pode compilar e linkar com OpenCV3 com o comando:

```
Windows ou Linux> compila nn_man -c -v3
```

Também pode compilar e linkar com OpenCV3 com o comando:

```
Linux> compila.sh nn_man
Linux> g++ -std=gnu++14 nn_man.cpp -o nn_man -fmax-errors=2 `pkg-config opencv4 --libs --cflags` -O3 -s
```

Executando o programa, obtemos:

```
Erros=1/60.   Pct=1.667%
```

O programa cometeu apenas um erro. Há 3 dados de teste localizados exatamente na fronteira entre as regiões “1” e “2”, de forma que essas instâncias podem ser classificadas como “1” ou

“2” dependendo do arredondamento numérico efetuado pelo algoritmo. Assim, o algoritmo pode errar entre 0 e 3 instâncias, sem que isto constitua nem mérito nem demérito.

O programa acima usou a rotina de vizinho mais próximo escrito “manualmente”. Mas isto não era necessário, pois dentro do OpenCV v2, há várias funções de aprendizado de máquina prontas, entre elas a classificação vizinho mais próximo.

```
 1 //nn_cv2.cpp
 2 #include <cekeikon.h>
 3 int main() {
 4   Mat_<float> ax; le(ax,"ax.txt");
 5   Mat_<float> ay; le(ay,"ay.txt");
 6   Mat_<float> qx; le(qx,"qx.txt");
 7   Mat_<float> qy; le(qy,"qy.txt");
 8   Mat_<float> qp;
 9   CvKNearest ind(ax,ay,Mat(),false,1);
10   ind.find_nearest(qx, 1, &qp);
11   int erros=0;
12   for (int i=0; i<qp.rows; i++)
13     if (qp(i)!=qy(i)) erros++;
14   printf("Erros=%d/%d.   Pct=%1.3f%%\n",erros,qp.rows,100.0*erros/qp.rows);
15 }
```

/home/hae/algpi/introaprend/iris2d/nn_cv2.cpp

Compile com OpenCV2:

```
$ compila iris_nn1 -c
```

Saída:

```
Erros=2/60.   Pct=3.333%
```

Em OpenCV v3 e v4, a forma de chamar a classificação vizinho mais próximo mudou. O programa abaixo mostra como é a chamada:

```
//nn_cv3.cpp
#include <cekeikon.h>
int main() {
  Mat_<float> ax; le(ax,"ax.txt");
  Mat_<float> ay; le(ay,"ay.txt");
  Mat_<float> qx; le(qx,"qx.txt");
  Mat_<float> qy; le(qy,"qy.txt");
  Mat_<float> qp;
  Ptr<ml::KNearest>  knn(ml::KNearest::create());
  knn->train(ax, ml::ROW_SAMPLE, ay);
  Mat_<float> dist;
  knn->findNearest(qx, 1, noArray(), qp, dist);
  int erros=0;
  for (int i=0; i<qp.rows; i++)
    if (qp(i)!=qy(i)) erros++;
  printf("Erros=%d/%d.   Pct=%1.3f%%\n",erros,qp.rows,100.0*erros/qp.rows);
}
```

/home/hae/algpi/introaprend/iris2d/nn_cv3.cpp

Para compilar com OpenCV3 com Cekeikon instalado:

```
$ compila iris_nn3 -c -v3
```

Para compilar com OpenCV4 sem Cekeikon:

```
$ g++ -std=gnu++14 $1.cpp -o $1 -fmax-errors=2 `pkg-config opencv4 --libs --cflags` -O3 -s
```

Saída:

```
Erros=1/60.   Pct=1.667%
```

O programa abaixo resolve o problema “Iris2D” usando classificador vizinho mais próximo da biblioteca SkLearn em Python. Além disso, o programa desenha a superfície de decisão no espaço dos atributos.

```
#iris_nn.py
import numpy as np
from sklearn import neighbors
import matplotlib.pyplot as plt

def le(nomearq):
  with open(nomearq,"r") as f:
    linhas=f.readlines()
  linha0=linhas[0].split()
  nl=int(linha0[0]); nc=int(linha0[1])
  a=np.empty((nl,nc),dtype=np.float32)
  for l in range(nl):
    linha=linhas[l+1].split()
    for c in range(nc):
      a[l,c]=np.float32(linha[c])
  return a

### main
ax=le("ax.txt"); ay=le("ay.txt")
qx=le("qx.txt"); qy=le("qy.txt")

vizinho = neighbors.KNeighborsClassifier(n_neighbors=1, weights="uniform", algorithm="brute")
vizinho.fit(ax,ay.ravel())

qp = vizinho.predict(qx)
erros=0;
for i in range(qp.shape[0]):
  if qp[i]!=qy[i]: erros+=1
print("Erros=%d/%d.   Pct=%1.3f%%\n"%(erros,qp.shape[0],100.0*erros/qp.shape[0]))

#https://machinelearningmastery.com/plot-a-decision-surface-for-machine-learning/
fig=plt.figure(figsize=(12, 12), dpi=80);
for i in range(ax.shape[0]):
  if   ay[i]==0: plt.plot(ax[i,0], ax[i,1], 'ro');
  elif ay[i]==1: plt.plot(ax[i,0], ax[i,1], 'go');
  elif ay[i]==2: plt.plot(ax[i,0], ax[i,1], 'bo');
for i in range(qx.shape[0]):
  if   qy[i]==0: plt.plot(qx[i,0], qx[i,1], 'rx');
  elif qy[i]==1: plt.plot(qx[i,0], qx[i,1], 'gx');
  elif qy[i]==2: plt.plot(qx[i,0], qx[i,1], 'bx');

minx = min( ax[:, 0].min()-1, qx[:, 0].min()-1 )
maxx = max( ax[:, 0].max()+1, qx[:, 0].max()+1 )
miny = min( ax[:, 1].min()-1, qx[:, 1].min()-1 )
maxy = max( ax[:, 1].max()+1, qx[:, 1].max()+1 )

gridx = np.arange(minx, maxx, 0.005)
gridy = np.arange(miny, maxy, 0.002)
xx, yy = np.meshgrid(gridx, gridy)

r1, r2 = xx.flatten(), yy.flatten()
r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))
grid = np.hstack((r1,r2))
yhat = vizinho.predict(grid)
zz = yhat.reshape(xx.shape)

# plot the grid of x, y and z values as a surface
plt.contourf(xx, yy, zz, cmap='Paired')
plt.tight_layout(); plt.show(); fig.savefig("iris_nn.png")
```

/home/hae/algpi/introaprend/iris2d/iris_nn.py

```
Erros=1/60.   Pct=1.667%
```

Olhando a superfície de decisão (figura abaixo), vemos que existe um dado de teste da classe “2” (virginica, azul) e dois dados de teste da classe “1” (versicolor, verde) na fronteira entre virginica e versicolor.

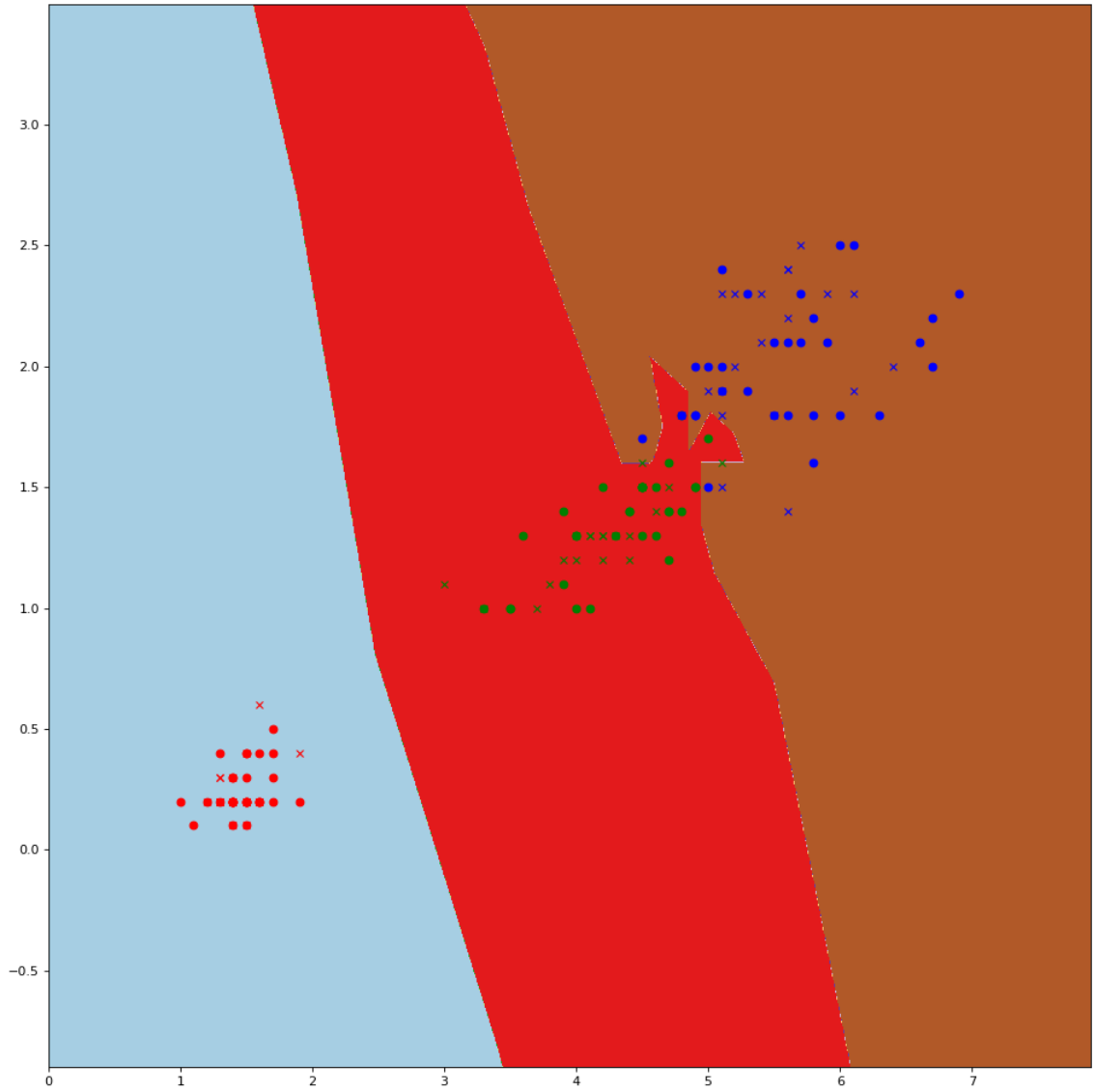


Figura: Superfície de decisão do classificador vizinho mais próximo, onde “o” são exemplos de treino e “x” são dados de teste. Marcas de cores RGB indicam respectivamente os exemplos das classes “0” (setosa), “1” (versicolor) e “2” (virginica). Existem 3 dados de teste na fronteira entre as classes “1” e “2”.

Exercício: Qual será a classificação dada pelo vizinho mais próximo para a instância (5.1; 1.6)? Explique.

Agora, vamos resolver o problema “Iris2D” usando árvore de decisão em Python/SkLearn:

```
#iris_dt2.py
import numpy as np
from sklearn import tree
import matplotlib.pyplot as plt

def le(nomearq):
  with open(nomearq,"r") as f:
    linhas=f.readlines()
  linha0=linhas[0].split()
  nl=int(linha0[0]); nc=int(linha0[1])
  a=np.empty((nl,nc),dtype=np.float32)
  for l in range(nl):
    linha=linhas[l+1].split()
    for c in range(nc):
      a[l,c]=np.float32(linha[c])
  return a

### main
ax=le("ax.txt"); ay=le("ay.txt")
qx=le("qx.txt"); qy=le("qy.txt")

arvore= tree.DecisionTreeClassifier()
arvore= arvore.fit(ax, ay)
qp=arvore.predict(qx)
erros=0;
for i in range(qp.shape[0]):
  if qp[i]!=qy[i]: erros+=1
print("Erros=%d/%d.   Pct=%1.3f%%\n"%(erros,qp.shape[0],100.0*erros/qp.shape[0]))

fig=plt.figure(figsize=(8,6))
tree.plot_tree(arvore,filled=True,fontsize=10)
plt.tight_layout(); plt.show(); fig.savefig("iris_dt2a.png")

#https://machinelearningmastery.com/plot-a-decision-surface-for-machine-learning/
fig=plt.figure(figsize=(12, 12), dpi=80); #plt.axis([0, 100, 0, 100])
for i in range(ax.shape[0]):
  if   ay[i]==0: plt.plot(ax[i,0], ax[i,1], 'ro');
  elif ay[i]==1: plt.plot(ax[i,0], ax[i,1], 'go');
  elif ay[i]==2: plt.plot(ax[i,0], ax[i,1], 'bo');
for i in range(qx.shape[0]):
  if   qy[i]==0: plt.plot(qx[i,0], qx[i,1], 'rx');
  elif qy[i]==1: plt.plot(qx[i,0], qx[i,1], 'gx');
  elif qy[i]==2: plt.plot(qx[i,0], qx[i,1], 'bx');

minx = min( ax[:, 0].min()-1, qx[:, 0].min()-1 )
maxx = max( ax[:, 0].max()+1, qx[:, 0].max()+1 )
miny = min( ax[:, 1].min()-1, qx[:, 1].min()-1 )
maxy = max( ax[:, 1].max()+1, qx[:, 1].max()+1 )

gridx = np.arange(minx, maxx, 0.001)
gridy = np.arange(miny, maxy, 0.0005)
xx, yy = np.meshgrid(gridx, gridy)

r1, r2 = xx.flatten(), yy.flatten()
r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))
grid = np.hstack((r1,r2))
yhat = arvore.predict(grid)
zz = yhat.reshape(xx.shape)

# plot the grid of x, y and z values as a surface
plt.contourf(xx, yy, zz, cmap='Paired')
plt.tight_layout(); plt.show(); fig.savefig("iris_dt2b.png")
```

/home/hae/algpi/introaprend/iris2d/iris_dt2.py

A saída obtida foi:

```
Erros=2/60.   Pct=3.333%
```

A árvore criada está abaixo. Uma árvore diferente é obtida cada vez que se executa o programa. Note como a árvore de decisão faz sucessivos cortes no espaço dos atributos. Verifique como as duas figuras abaixo (árvore e superfície de decisão) estão relacionadas.

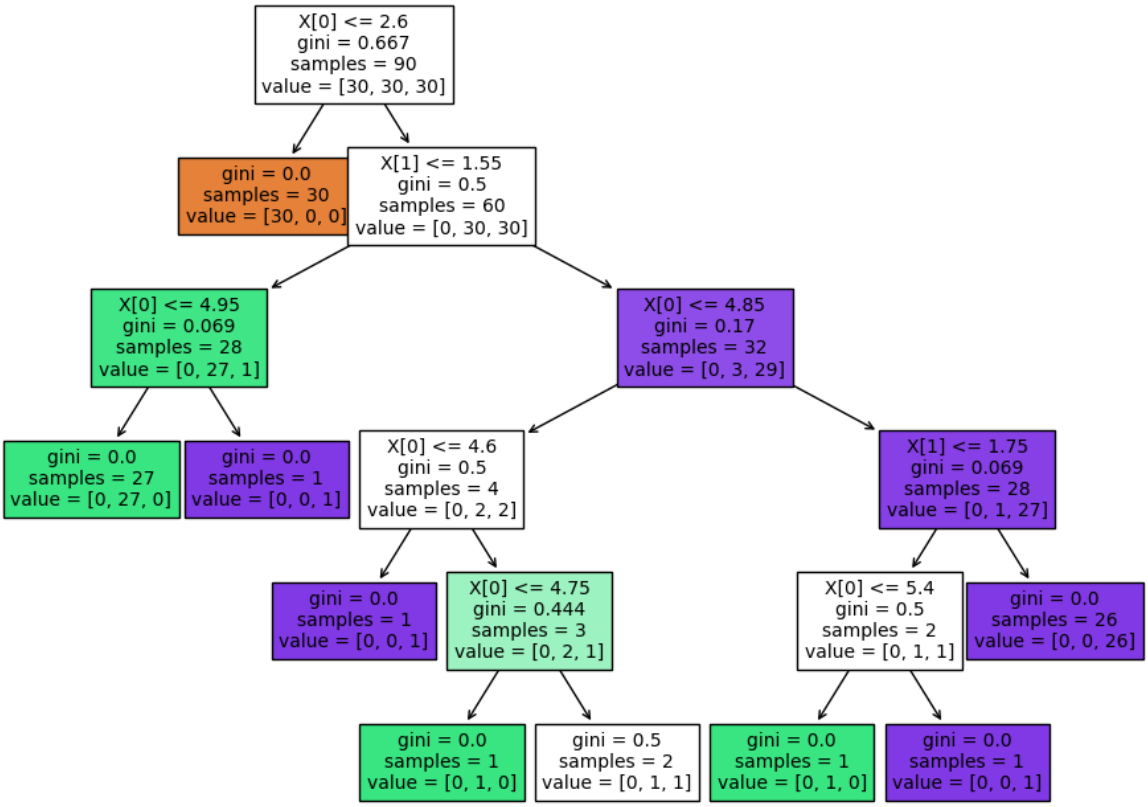


Figura: Árvore de decisão criada.

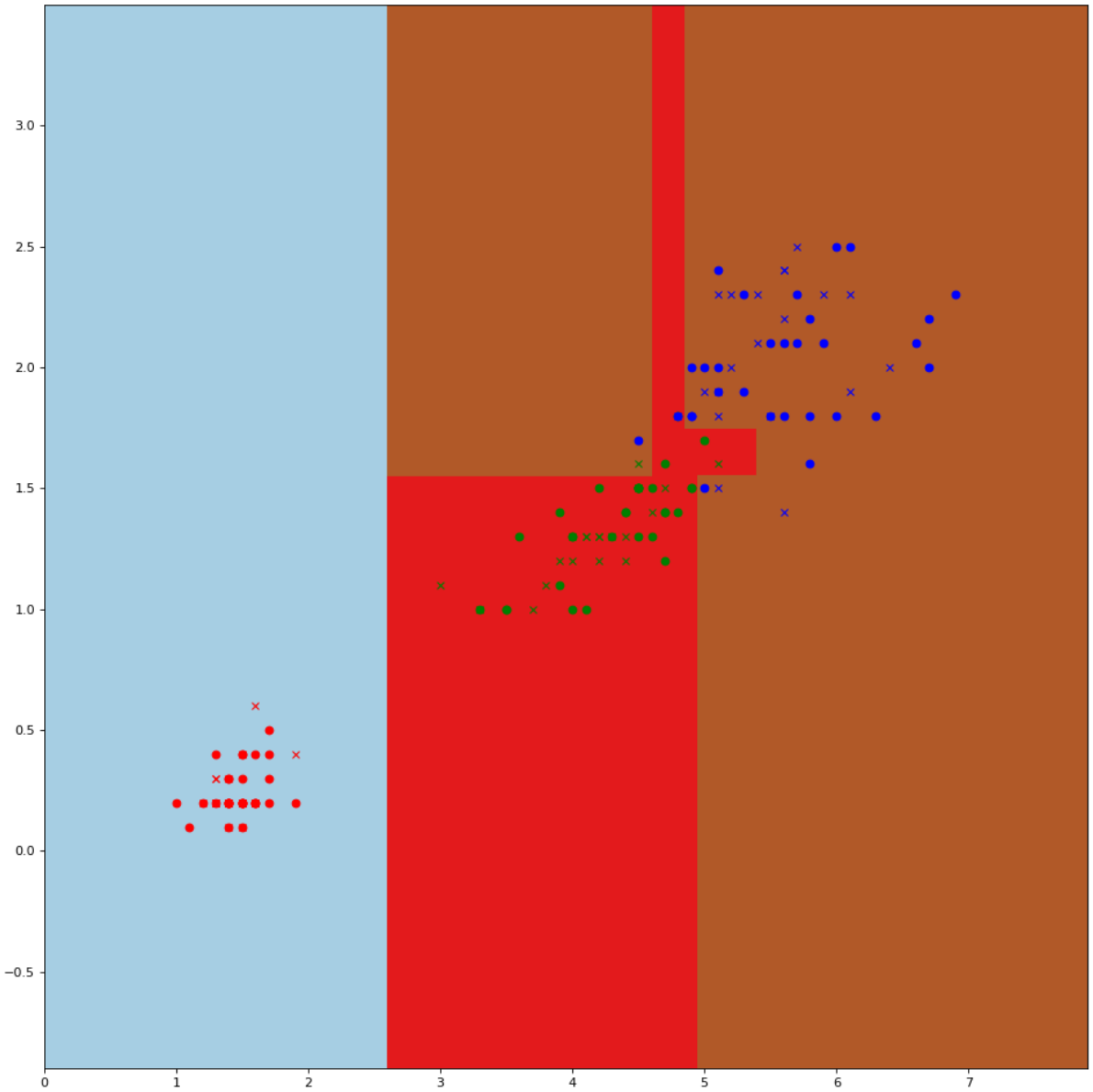


Figura: Superfície de decisão da árvore de decisão, onde “o” são exemplos de treino e “x” são dados de teste. Marcas RGB indicam respectivamente os exemplos 0 (setosa), 1 (versicolor) e 2 (virginica). A cor de fundo representa a divisão do espaço de atributos.

Exercício: Qual será a classificação dada pela árvore de decisão para a instância (5.1, 1.6)? Mostre a sequência de nós da árvore visitados ao classificar essa instância.

Até agora, vimos problemas onde os atributos eram valores reais contínuos (kg, cm, etc) e os rótulos eram discretos (A, B, C, setosa, versicolor, virginica). Existem problemas de aprendizado de máquina onde os atributos de entrada são discretos assim como problemas onde o espaço de saída é contínua.

Banknote

Nos sites:
https://archive.ics.uci.edu/ml/datasets/banknote+authentication OU
https://machinelearningmastery.com/standard-machine-learning-datasets item (5)
há o arquivo *data_banknote_authentication.txt* com 4 características extraídas de notas de banco verdadeiras (classe 0) e falsas (classe 1). Peguei as linhas pares desse arquivo (começando da linha 0) para treino e as ímpares para teste, gerando os arquivos *ax.txt*, *ay.txt*, *qx.txt* e *qy.txt*, conforme a convenção adotada nesta apostila. As primeiras linhas desses arquivos são:

data_banknote_authentication.txt	ax.txt	ay.txt	qx.txt	qy.txt
3.6216,8.6661,-2.8073,-0.44699,0 4.5459,8.1674,-2.4586,-1.4621,0 3.866,-2.6383,1.9242,0.10645,0 3.4566,9.5228,-4.0112,-3.5944,0	686 4 3.6216 8.6661 -2.8073 -0.44699 3.866 -2.6383 1.9242 0.10645	686 1 0.0 0.0	686 4 4.5459 8.1674 -2.4586 -1.4621 3.4566 9.5228 -4.0112 -3.5944	686 1 0.0 0.0

Esses arquivos estão em:
http://www.lps.usp.br/hae/apostila/banknote.zip

1) Escreva um programa C++ ou Python que classifica as instâncias *qx* usando as amostras de treinamento (*ax*, *ay*) e imprime a taxa de erro obtida usando o método vizinho mais próximo. Pode usar a implementação manual do vizinho mais próximo, rotina do OpenCV2, OpenCV3 ou sklearn.
Resultado esperado: Comete 1 erro, independentemente da linguagem e biblioteca utilizada.

2) O mesmo usando a árvore de decisão.
Resultado esperado: Comete 18 erros em OpenCV2. Comete 8 erros em Python/sklearn.

Opcional: Use algum método de aprendizado de máquina diferente do vizinho mais próximo e da árvore de decisão para conseguir taxa de erro menor ainda.
[Solução em ~/algpi/introaprend/banknote e “apostila privada” aprend.odt]

Noisynote

Vamos inserir 3 atributos que são simplesmente ruídos, para verificar se os algoritmos de aprendizagem funcionam mesmo na presença desses atributos. Nos arquivos "?x.txt" abaixo, as últimas 3 colunas são ruído.

ax.txt	ay.txt	qx.txt	qy.txt
686 7 3.62 8.67 -2.81 -0.45 7.72 -20.09 -19.15 3.87 -2.64 1.92 0.11 -11.57 -14.66 16.60	686 1 0.0 0.0	686 7 4.55 8.17 -2.46 -1.46 4.21 -5.35 -7.17 3.46 9.52 -4.01 -3.59 17.75 -16.56 -18.02	686 1 0.0 0.0

Esses arquivos estão em:

http://www.lps.usp.br/hae/apostila/noisynote.zip

Executando vizinho mais próximo implementado manualmente, vizinho mais próximo de OpenCV3 ou Python/SkLearn, obtive 193 erros (de um total de 686), ou 28.13% de erros.

Executando vizinho mais próximo de OpenCV2, obtive 184/686 erros ou 26.822%.

Executando decision-tree de Python/SkLearn, obtive 12/686 ou 1.75% de erro.

Executando decision-tree de OpenCV2 (selecionando bons parâmetros), obtive 18/686 ou 2.624% de erro.

Assim, fica claro que quando há atributos ruidosos, decision-tree funciona muito melhor que vizinho mais próximo.

Tabela X: Resultados que obtive resolvendo o problema "noisynoite".

/home/hae/algpi/introaprend/noisynote

Método	Número de erros em 686 testes	Porcentagem de erro
Viz+Px manual, OpenCV3, SkLearn	193	28.13%
Viz+Px OpenCV2	184	26.822%
Decision-tree SkLearn	12	1.75%
Decision-tree OpenCV2	18	2.624%

[PSI5790, aula 4, lição de casa #2 (de 2)] Resolva o problema "noisynote" usando os algoritmos de vizinho mais próximo e árvore de decisão.

[PSI5790, aula 4 parte 2. Fim.]