Projecto 2
Árvores de decisão

Neste projecto vamos estudar o comportamento de árvores de decisão. Para um dado conjunto de dados iremos inferir uma árvore de decisão que explica os dados.

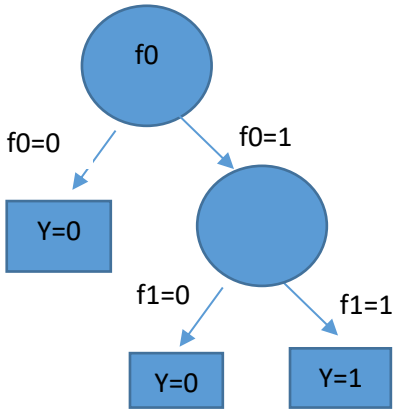
Como exemplo, considere os dados seguintes. Os dados estão num array numpy *D* onde as linhas são os exemplos e as colunas são as features (3 neste caso). As classificações estão no array *Y*. Consideramos os casos de variáveis binárias.

	f0	f1	Class (Y)
Ex1	0	0	False
Ex2	0	1	False
Ex3	1	0	False
Ex4	1	1	True

```
D = array([[0, 0],
           [0, 1],
           [1, 0],
           [1, 1]])
Y = array([0, 0, 0, 1])
```

Uma árvore de decisão é, neste caso, uma árvore binária, onde cada nó corresponde a uma só feature e cada ramo corresponde ao caso onde a feature é 0 (Falso) ou 1 (Verdadeiro). Podemos codificar a árvore com uma lista de listas onde o primeiro elemento é o índice para a feature, o segundo é o ramo para a feature negativa, e o terceiro elemento corresponde ao caso positivo.

T = [0, 0, [1, 0, 1]]



(12 val) Inferência de árvores de decisão

Neste projecto vamos desenvolver uma função para fazer a inferência de árvores de decisão a partir de dados. A função será:

```
def createdecisiontree(D,Y, noise=False):
    (a desenvolver)
        return tree
T = createdecisiontree(D,Y)
```

Podemos usar o método descrito na Figura 18 do capítulo 18 do livro. Podemos considerar o caso onde não há ruído, i.e. no conjunto de treino é o mesmo do que o conjunto de teste é cada exemplo só pode ter uma classificação.

Uma função que aceita uma árvore de decisão e um conjunto de dados e devolve a classificação é fornecido:

```
def classify(T,data):
    (...)
Yp = classify(T,D)
```

Exemplo de utilização:

```
Input

print("dataset > ", idataset, "\nD", D, "\nY", Y)
T = createdecisiontree(D,Y)
Yp = classify(T,D)
err = np.mean(np.abs(Yp-Y))
print("tree > ", T, "\nprediction >", Yp,"\n   correct >",Y,"\n    errors >
", err, "tree length", l)

Output:

dataset >  1
D [[0 0]
 [0 1]
 [1 0]
 [1 1]]

Y [0 0 0 1]

tree >  [0, 0, [1, 0, 1]]
```

(3 val) Árvores mais pequenas

Podemos notar ao analisar os resultados que por vezes não obtemos a árvore mais simples.

Podemos observar isso neste exemplo:

```
D = array([[0, 0, 0],
           [0, 0, 1],
           [0, 1, 0],
           [0, 1, 1],
           [1, 0, 0],
           [1, 0, 1],
           [1, 1, 0],
           [1, 1, 1]])
```

```
Y = array([0, 1, 1, 0, 0, 1, 1, 0])
```

Árvore inferida:

T > [0, [1, [2, 0, 1], [2, 1, 0]], [1, [2, 0, 1], [2, 1, 0]]]

Árvore mais curta:

T2 > [1, [2, 0, 1], [2, 1, 0]]

Podemos avaliar o tamanha da árvore pelo número de nós, ou de uma forma aproximada pelo número de caracteres na string. A árvore original tem tamanha 57 a árvore mais comprimida tem tamanha 25, ambas com erro zero.

Como podemos melhorar/adaptar o método usado para reduzir o tamanha das árvores de decisão, como no exemplo de cima?

A função "`createdecisiontree`" deverá ser alterada para reduzir o tamanho das árvores.

Sugestões: capítulo 18.3.5, 18.4, https://scikit-learn.org/stable/modules/tree.html#tree

(3 val) Ruído.

No mundo real os dados são ruidosos. Por vezes, o mesmo exemplo pode aparecer mais do que uma vez no conjunto de dados com classificações diferentes. Outra maneira de ver isto é que o conjunto usados para treino podem não ser exactamente os mesmos do que os os dados onde vamos usar o sistema.

Considerando o seguinte conjunto de treino (D,Y) e conjunto de teste (Dt,Yt) onde os dois primeiros exemplos têm classificações diferentes no conjunto de treino e conjunto de teste.

D= [[0 0 0] [0 0 1] [0 1 0] [0 1 1] [1 0 0] [1 0 1] [1 1 0] [1 1 1]]	Dt= [[0 0 0] [0 0 1] [0 1 0] [0 1 1] [1 0 0] [1 0 1] [1 1 0] [1 1 1]]
Y= [1 0 0 1 0 1 0 1]	Yt= [0 1 0 1 0 1 0 1]

Considerando as 2 árvores seguintes:

T > [2, [0, [1, 1, 0], 0], [0, [1, 0, 1], 1]]

T2> [2, 0, 1]

T2 tem erro 0 no teste mas não no treino, T tem erro 0 no treino mas não no teste. Qual é a melhor? Notar que o erro obtido no conjunto de teste é o que nós queremos minimizar apesar de de não termos acesso a eles no treino.

Altere a função inicial para lidar com esta situação.

Os casos que vão ser avaliados desta forma irão ser chamados usando um parâmetro extra.

createdecisiontree(D,Y, noise=True)

Sugestões: capítulo 18.3.5, 18.4, https://scikit-learn.org/stable/modules/tree.html#tree

(2 val) Relatório

O relatório deverá ter uma explicação dos problemas encontrados no algoritmo base (problemas ao lidar com ruído, não encontrar a árvore mais pequena, outros), quais as possíveis soluções para esses problemas, descrição da descrição encontrada, análise crítica dos resultados. Limite de páginas 2 (tamanho 12), sem limite para gráficos, tabelas, figuras e referências.

Avaliação

A pontuação consiste:

(8pts) execução correcta nos exemplos fornecidos

(3pts) algoritmo para reduzir tamanha das árvores

(3pts) algoritmo para lidar com o ruído

(4pts) execução correcta noutros exemplos

(2pts) relatório

Penalizações :

(-2pts) dados/informação em falta / formato errado;

(-5pts) erros sintáticos;

Ficheiros

Deverá ser entregue um ficheiro com nome alxxx.py/tgxxx.py (onde *al* corresponde a Alameda, *tg* corresponde ao Taguspark, e xxx é o número do grupo com 3 dígitos). O ficheiro deverá ter a identificação (nome + número e o número do grupo incluindo al/tg). Deverá ser entregue o relatório em formato .pdf (alxxx.odf/tgxxx.pdf). Submeter no fénix num só ficheiro alxxx.zip/tgxxx.zip

datasetstreelearning.py – ficheiro que contem vários dados para teste

testdecisiontrees.py – ficheiro que realiza os testes e faz a correcção. Não alterar.

Condições

Grupos de 2 elementos

Prazo: 07/12/2020 a submeter no fenix

Se os dois elementos do grupo não contribuíram de forma igual deverá ser declarado.

Poderá haver discussões do projecto.

*** TODO O CÓDIGO DEVE SER ORIGINAL ***