



INSTITUTO FEDERAL MINAS GERAIS (IFMG) - CAMPUS BAMBUÍ
Mineração de Dados
Prof. Marcos Roberto Ribeiro

Lista de Exercícios 05

Exercício 1:

Considere o conjunto de dados mostrado na Figura 1.

Aparência	Temperatura	Umidade	Vento	Jogo
sol	quente	alta	falso	não
sol	quente	alta	verdade	não
encoberto	quente	alta	falso	sim
chuvoso	agradável	alta	falso	sim
chuvoso	frio	normal	falso	sim
chuvoso	frio	normal	verdade	não
encoberto	frio	normal	verdade	sim
sol	agradável	alta	falso	não
sol	frio	normal	falso	sim
chuvoso	agradável	normal	falso	sim
sol	agradável	normal	verdade	sim
encoberto	agradável	alta	verdade	sim
encoberto	quente	normal	falso	sim
chuvoso	agradável	alta	verdade	não

Figura 1: Dados para determinar se haverá jogo

- Construa uma árvore de decisão para essa base de dados.
- Calcule a precisão desta árvore considerando todo o conjunto de treinamento.
- Prepare a base de dados para o Weka, execute os algoritmos de árvore de decisão do Weka e compare com a árvore desenvolvida.

Exercício 2:

Desenhe uma árvore de decisão para um conjunto de dados contendo quatro atributos binários p , q , r e s . A classe dos registros é o resultado da fórmula $A = (p \wedge q \wedge r \wedge s)$. Quando essa árvore pode classificar um registro de forma incorreta?

Exercício 3:

Explique como é o processo de criação de uma árvore de decisão.

Exercício 4:

Para que serve a medida de entropia? Qual a vantagem de usá-la na criação de árvores de decisão?

Exercício 5:

Cite duas vantagens da árvore de decisão.

Exercício 6:

Considere a classe de um classificador genérico do Apêndice A. Implemente uma de árvore de decisão herdando de **Classifier**. Verifique a precisão de sua implementação usando os dados da Figura 1, considere o conjunto de treinamento completo e a validação cruzada com 10 partições. Durante a implementação implemente os métodos abstratos da classe pai e considere a criação dos seguintes métodos:

- `_split(data, att)`: particiona os registros de **data** de acordo com os valores do atributo **att**;
- `_entropy(data)` : Calcula a entropia dos registros de **data**;
- `_impurity(att, div_dict)` : Calcula a impureza do particionamento **div_dict** (criado usando **att**), o particionamento pode ser um dicionário **valor: registros com att=valor**;
- `_get_best_att(data)`: Seleciona atributo com melhor ganho de informação para particionar os registros de **data**;
- `_build_tree(data)` : Cria a árvore (recursivamente) sobre os registros de **data**;
- `print_tree()` : Imprime a estrutura da árvore;

Referências

TAN, P.-N.; STEINBACH, M.; KUMAR, V. **Introdução ao data mining**: mineração de dados. Rio de Janeiro: Ciência Moderna, 2009.

Apêndice A Classificador genérico

```
1  #!/usr/bin/python3
2  # -*- coding: utf-8 -*-
3
4  import pandas as pd
5  import logging
6  from abc import abstractmethod
7
8
9  LOG = 'classifier.log'
10
11
12  class Classifier():
13      '''
14      Classifier template class
15      '''
16      def __init__(self, csv_file, debugging=False):
17          # Flag de depuração
18          self._debugging = debugging
19          # Configura log
20          self._config_log()
21          # Lê arquivo CSV para _data
22          self._data = pd.read_csv(csv_file, skipinitialspace=True)
23          # Pega último atributo para classe
24          self.class_att = str(self._data.columns[-1])
25          # Demais atributos são atributos de dados
26          self.att_list = list(self._data.columns[:-1])
27
28      @abstractmethod
29      def fit(self):
30          '''
31          Treinamento do classificador
32          '''
33          pass
34
35      @abstractmethod
36      def classify_record(self, record):
37          '''
38          Classifica um registro
39          '''
40          pass
41
42      def _debug(self, msg, *args, **kwargs):
43          '''
44          Exibe mensagens de debug
45          '''
46          self._log.debug(msg, *args, **kwargs)
47
48      def _config_log(self):
49          '''
50          Configura o log
51          '''
52          # Cria log
```

```

53     self._log = logging.getLogger(LOG)
54     # Formato de mensagens
55     str_format = '%(levelname)s - %(message)s'
56     log_format = logging.Formatter(str_format)
57     # Arquivo
58     file_handler = logging.FileHandler(LOG)
59     file_handler.setFormatter(log_format)
60     # Console
61     console_hanler = logging.StreamHandler()
62     console_hanler.setFormatter(log_format)
63     self._log.addHandler(file_handler)
64     self._log.addHandler(console_hanler)
65     # Nível de log
66     if self._debugging:
67         self._log.setLevel(logging.DEBUG)
68     else:
69         self._log.setLevel(logging.INFO)
70
71     def disable_debug(self):
72         """
73         Desabilita o debug
74         """
75         self._log.setLevel(logging.INFO)
76
77     def precision(self, test_data):
78         """
79         Calcula a precisão considerando os dados de test_data
80         """
81         # Contagem de erros
82         errors = 0
83         # Para cada registro de dado
84         for _, rec in test_data.iterrows():
85             # Classifica o registro
86             class_pre = self.classify_record(rec)
87             # Compara com a classe correta do registro
88             if class_pre != rec[self.class_att]:
89                 # Se for diferente conta o erro
90                 errors += 1
91         # Retorna a porcentagem de acertos
92         return (len(test_data) - errors) / len(test_data)
93
94     def training_precision(self):
95         """
96         Calcula a precisão usando todo o conjunto de treinamento
97         """
98         return self.precision(self._data)
99
100     def k_fold_precision(self, k):
101         """
102         Calcula a precisão usando validação cruzada
103         """
104         # Faz cópia dos dados originais
105         bkp_data = self._data.copy(deep=True)
106         # Tamanho de cada partição
107         fold_len = len(bkp_data) // k

```

```
108     # Soma da precisão para fazer a média
109     precision = 0
110     # Para cada partição
111     for cont in range(k):
112         # Calcula o início e fim dos dados de teste
113         fold_start = cont*fold_len
114         fold_end = (cont+1)*fold_len
115         # Seleciona os dados de teste
116         test_data = bkp_data[fold_start:fold_end]
117         # Os dados de treinamento são os demais dados
118         train_data = bkp_data.drop([fold_start, fold_end-1])
119         # Atribui os dados de teste ao classificador
120         self._data = train_data
121         # Faz o treinamento
122         self.fit()
123         # Soma a precisão
124         precision += self.precision(test_data)
125     # Retorna a média das precisões
126     return precision / k
```