Estimação, Deteção e Análise II

01 – Fundamentos Gerais

Regras de Associação (APRIORI)

Árvores de decisão (classificação e regressão)

Regressão linear simples e múltipla

Configuração Inicial

1. No Anaconda, criar um *environment* chamado EDA2
2. No PyCharm, criar um projeto chamado EDA2 e associá-lo ao *environment* criado. Neste link encontra-se um guia de como associar um projeto do PyCharm a um *environment* do Anaconda.
3. Instalar, no *environment*, os packages necessários para a resolução dos exercícios

Regras de Associação (APRIORI)

O objetivo deste exercício é reproduzir o "Laboratório 6 – Data Mining: Association Rules" resolvido com o Weka. Num primeiro momento, deve relembrar os resultados desse exercício.

```
Best rules found:

 1. biscuits=t frozen foods=t fruit=t total=high 788 ==> bread and cake=t 723    <conf:(0.92)> lift:(1.27) lev:(0.03) [155] conv:(3.35)
 2. baking needs=t biscuits=t fruit=t total=high 760 ==> bread and cake=t 696    <conf:(0.92)> lift:(1.27) lev:(0.03) [149] conv:(3.28)
 3. baking needs=t frozen foods=t fruit=t total=high 770 ==> bread and cake=t 705    <conf:(0.92)> lift:(1.27) lev:(0.03) [150] conv:(3.27)
 4. biscuits=t fruit=t vegetables=t total=high 815 ==> bread and cake=t 746    <conf:(0.92)> lift:(1.27) lev:(0.03) [159] conv:(3.26)
 5. party snack foods=t fruit=t total=high 854 ==> bread and cake=t 779    <conf:(0.91)> lift:(1.27) lev:(0.04) [164] conv:(3.15)
 6. biscuits=t frozen foods=t vegetables=t total=high 797 ==> bread and cake=t 725    <conf:(0.91)> lift:(1.26) lev:(0.03) [151] conv:(3.06)
 7. baking needs=t biscuits=t vegetables=t total=high 772 ==> bread and cake=t 701    <conf:(0.91)> lift:(1.26) lev:(0.03) [145] conv:(3.01)
 8. biscuits=t fruit=t total=high 954 ==> bread and cake=t 866    <conf:(0.91)> lift:(1.26) lev:(0.04) [179] conv:(3)
 9. frozen foods=t fruit=t vegetables=t total=high 834 ==> bread and cake=t 757    <conf:(0.91)> lift:(1.26) lev:(0.03) [156] conv:(3)
10. frozen foods=t fruit=t total=high 969 ==> bread and cake=t 877    <conf:(0.91)> lift:(1.26) lev:(0.04) [179] conv:(2.92)
```

Analisar o funcionamento do algoritmo APRIORI em Python em https://www.javatpoint.com/apriori-algorithm-in-machine-learning

No projeto EDA2 do PyCharm, criar um ficheiro chamado decisionrules_supermarket.py para guardar o código para as seguintes instruções:

1. Instalar o package apyori

2. Ler o ficheiro supermarket.csv que se encontra no Moodle:

```
import pandas as pd  # To read data
data = pd.read_csv('caminho_fich/supermarket.csv')
```

Neste caso, a leitura do ficheiro é feita com recurso à função read_csv da biblioteca pandas. A documentação desta função pode ser encontrada neste link e a documentação geral da biblioteca pandas pode ser encontrada neste link.

3. Converter o dataset "data" para o formato esperado pelo algoritmo:

```
records = []
for index, row in data.iterrows():
    record = []
    for c in data.columns:
        if row[c] == 't':
            record.append(c)
        if c=="total":
            record.append("total_"+row[c])
    records.append(record)
```

Excerto da lista (records[0], correspondente à transação registada na primeira linha do csv):

```
['baby needs', 'bread and cake', 'baking needs', 'juice-sat-cord-ms', 'biscuits',
'canned vegetables', 'cleaners-polishers', 'coffee', 'sauces-gravy-pkle',
'confectionary', 'dishcloths-scour', 'frozen foods', 'razor blades', 'party snack
foods', 'tissues-paper prd', 'wrapping', 'mens toiletries', 'cheese', 'milk-cream',
'margarine', 'small goods', 'fruit', 'vegetables', 'department122', '750ml white nz',
'total_high']
```

4. Executar o algoritmo e verificar o resultado

```
from apyori import apriori

rules = apriori(records, min_support=0.1, min_confidence=0.9, min_length=2, min_lift=1.25)
listrules = list(rules)
```

5. Visualizar o resultado

```
for item in listrules:
    print(item)
```

Excerto do resultado (primeira regra encontrada):

```
RelationRecord(items=frozenset({'bread and cake', 'total_high', 'baking needs',
'beef'}), support=0.13140263669764427, ordered_statistics = [OrderedStatistic(items_base
= frozenset({'total_high', 'baking needs', 'beef'}), items_add = frozenset({'bread and
cake'}), confidence=0.9007407407407408, lift=1.2515697920142366)])
```

Este formato é difícil de ler. Podemos "partir" o resultado de forma a ficar mais legível:

```
RelationRecord(
        items=frozenset({'bread and cake', 'total_high', 'baking needs', 'beef'}),
        support=0.13140263669764427,
        ordered_statistics = [OrderedStatistic(
                items_base = frozenset({'total_high', 'baking needs', 'beef'}),
                items_add = frozenset({'bread and cake'}),
                confidence=0.9007407407407408,
                lift=1.2515697920142366)]     )
```

Ou, para simplificar ainda mais, podemos usar o seguinte código para formatar o resultado

```
for item in listrules:
    pair = item[0]
    items = [x for x in pair]
    ant = str(list(item[2][0][0]))[1:-1]
    cons = str(list(item[2][0][1]))[1:-1]
    print("Rule: {" + ant + "} -> {" + cons + "}")
    print("Support: " + str(item[1]))
    print("Confidence: " + str(item[2][0][2]))
    print("Lift: " + str(item[2][0][3]))
    print("====================================")
```

Excerto do resultado (primeira regra encontrada, reformatada):

```
Rule: {'beef', 'baking needs', 'total_high'} -> {'bread and cake'}
Support: 0.13140263669764427
Confidence: 0.9007407407407408
Lift: 1.2515697920142366
```

6. Interpretar o resultado

A primeira regra diz que as transações que contêm 'beef', 'baking needs' e 'total_high' deverão provavelmente ter também 'bread and cake'. Esta regra tem:

- Support de 13%: Os items 'beef', 'baking needs', 'total_high' e 'bread and cake' aparecem juntos em 13% das transações
- Confidence de 90%: Transações com 'beef', 'baking needs' e 'total_hig têm uma probabilidade de 90% de também conter 'bread and cake'
- Lift de 1.25%: como o Lift é maior que 1, o antecedente e o consequente aparecem mais frequentemente juntos do que esperado, ou seja: a ocorrência do antecedente tem um efeito positivo na ocorrência do consequente

Árvores de decisão

Classificação

O objetivo deste exercício é reproduzir o "Laboratório 4 – Data Mining: Classification #1" resolvido com o Weka. Num primeiro momento, deve relembrar os resultados desse exercício.

Resultados do modelo:

```
J48 pruned tree
------------------

feathers = FALSE
|   milk = TRUE: mammal (41.0)
|   milk = FALSE
|   |   backbone = TRUE
|   |   |   fins = FALSE
|   |   |   |   tail = FALSE: amphibian (3.0)
|   |   |   |   tail = TRUE: reptile (6.0/1.0)
|   |   |   fins = TRUE: fish (13.0)
|   |   backbone = FALSE
|   |   |   airborne = FALSE
|   |   |   |   predator = TRUE: invertebrate (8.0)
|   |   |   |   predator = FALSE
|   |   |   |   |   legs <= 2: invertebrate (2.0)
|   |   |   |   |   legs > 2: insect (2.0)
|   |   |   airborne = TRUE: insect (6.0)
feathers = TRUE: bird (20.0)

Number of Leaves  :  9

Size of the tree :  17
```

Avaliação do modelo

```
Correctly Classified Instances          93               92.0792 %
Incorrectly Classified Instances         8                7.9208 %
Kappa statistic                          0.8955
Mean absolute error                      0.0225
Root mean squared error                  0.14
Relative absolute error                 10.2478 %
Root relative squared error             42.4398 %
Total Number of Instances              101

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               1,000    0,000    1,000      1,000   1,000      1,000    1,000     1,000     mammal
               1,000    0,011    0,929      1,000   0,963      0,958    0,994     0,929     fish
               1,000    0,000    1,000      1,000   1,000      1,000    1,000     1,000     bird
               0,800    0,033    0,727      0,800   0,762      0,735    0,986     0,812     invertebrate
               0,625    0,032    0,625      0,625   0,625      0,593    0,920     0,677     insect
               0,750    0,000    1,000      0,750   0,857      0,862    0,872     0,760     amphibian
               0,600    0,010    0,750      0,600   0,667      0,656    0,793     0,420     reptile
Weighted Avg.  0,921    0,008    0,922      0,921   0,920      0,914    0,976     0,908

=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
 41  0  0  0  0  0  0 |  a = mammal
  0 13  0  0  0  0  0 |  b = fish
  0  0 20  0  0  0  0 |  c = bird
  0  0  0  8  2  0  0 |  d = invertebrate
  0  0  0  3  5  0  0 |  e = insect
  0  0  0  0  0  3  1 |  f = amphibian
  0  1  0  0  1  0  3 |  g = reptile
```

No projeto EDA2 do PyCharm, criar um ficheiro chamado decisiontree_zoo.py para guardar o código para as seguintes instruções:

1. Instalar o package sklearn

2. Ler o ficheiro zoo.csv que se encontra no Moodle:

```
import pandas as pd  # To read data
data = pd.read_csv(' caminho_fich/zoo.csv')
```

3. Definir quais são as variáveis independentes (X) e dependente (y)

```
X = data[['hair', 'feathers', 'eggs', 'milk', 'airborne', 'aquatic', 'predator',
        'toothed', 'backbone', 'breathes', 'venomous', 'fins', 'legs', 'tail',
        'domestic', 'catsize']]
Y = data[['type']]
```

4. Criar e fazer fit da árvore de decisão

```
from sklearn import tree
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, Y)
```

5. Ver o modelo criado

```
from sklearn.tree import export_text
r = export_text(clf, feature_names=['hair', 'feathers', 'eggs', 'milk', 'airborne',
        'aquatic', 'predator', 'toothed', 'backbone', 'breathes', 'venomous', 'fins',
        'legs', 'tail', 'domestic', 'catsize'])
print(r)
```

Resultado

```
|--- milk <= 0.50
|   |--- feathers <= 0.50
|   |   |--- fins <= 0.50
|   |   |   |--- backbone <= 0.50
|   |   |   |   |--- airborne <= 0.50
|   |   |   |   |   |--- predator <= 0.50
|   |   |   |   |   |   |--- legs <= 3.00
|   |   |   |   |   |   |   |--- class: invertebrate
|   |   |   |   |   |   |--- legs >  3.00
|   |   |   |   |   |   |   |--- class: insect
|   |   |   |   |   |--- predator >  0.50
|   |   |   |   |   |   |--- class: invertebrate
|   |   |   |   |--- airborne >  0.50
|   |   |   |   |   |--- class: insect
|   |   |   |--- backbone >  0.50
|   |   |   |   |--- aquatic <= 0.50
|   |   |   |   |   |--- class: reptile
|   |   |   |   |--- aquatic >  0.50
|   |   |   |   |   |--- breathes <= 0.50
|   |   |   |   |   |   |--- class: reptile
|   |   |   |   |   |--- breathes >  0.50
|   |   |   |   |   |   |--- class: amphibian
|   |   |--- fins >  0.50
|   |   |   |--- class: fish
|   |--- feathers >  0.50
|   |   |--- class: bird
|--- milk >  0.50
|   |--- class: mammal
```

6. Desenhar a árvore de decisão

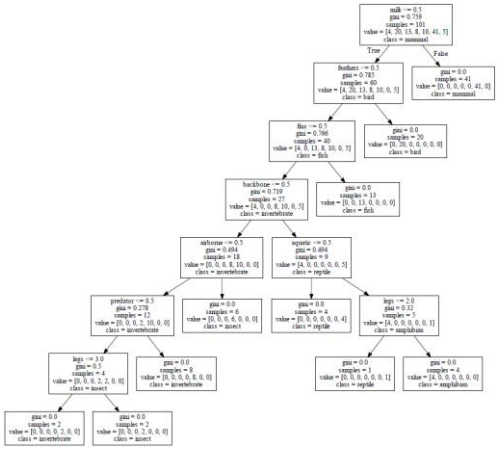
```
dot_data = tree.export_graphviz(clf, out_file=None,
                                feature_names=['hair', 'feathers', 'eggs', 'milk',
                                'airborne', 'aquatic', 'predator','toothed', 'backbone',
                                'breathes', 'venomous', 'fins', 'legs', 'tail',
                                'domestic', 'catsize'],
                                class_names=clf.classes_)

import pydotplus
graph = pydotplus.graph_from_dot_data(dot_data)

from IPython.display import Image

Image(graph.create_png())
graph.write_pdf("dt.pdf")
```

7. Verificar o resultado guardado no ficheiro dt.pdf dentro do projeto EDA2



Nota: A árvore criada é ligeiramente diferente da criada com o Weka, o que se deve à aleatoriedade intrínseca ao algoritmo de árvores de decisão.

8. Avaliar o modelo utilizando cross-validation (10-fold)

```
from sklearn import tree
clf = tree.DecisionTreeClassifier()

from sklearn.model_selection import cross_val_predict
y_pred = cross_val_predict(clf, X, Y, cv=10)

from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(Y, y_pred))
print(classification_report(Y, y_pred))
from sklearn.model_selection import cross_validate
from statistics import mean
cv_results = cross_validate(clf, X, Y, cv=10)
print("Accuracy:", mean(cv_results['test_score']))
```

Resultados:

```
[[ 3  0  0  0  0  0  1]
 [ 0 20  0  0  0  0  0]
 [ 0  0 13  0  0  0  0]
 [ 0  0  0  7  1  0  0]
 [ 0  0  0  1  9  0  0]
 [ 0  0  0  0  0 41  0]
 [ 1  0  0  1  0  0  3]]
              precision    recall  f1-score   support

   amphibian       0.75      0.75      0.75         4
        bird       1.00      1.00      1.00        20
        fish       1.00      1.00      1.00        13
      insect       0.78      0.88      0.82         8
invertebrate       0.90      0.90      0.90        10
      mammal       1.00      1.00      1.00        41
     reptile       0.75      0.60      0.67         5

    accuracy                           0.95       101
   macro avg       0.88      0.88      0.88       101
weighted avg       0.95      0.95      0.95       101

Accuracy: 0.93
```

Nota: Como a árvore criada é ligeiramente diferente da criada com o Weka, o seu desempenho também é diferente

Regressão

O objetivo deste exercício é reproduzir o "Laboratório 1 – Data Mining: Regressão" resolvido com o Weka com o modelo LinearRegression, mas com árvores de decisão para regressão. Num primeiro momento, deve relembrar os resultados desse exercício.

Resultados do modelo:

```
Linear Regression Model

SellingPrice =

    -26.6882 * HouseSize +
      7.0551 * LotSize +
  43166.0767 * Bedrooms +
  42292.0901 * Bathroom +
 -21661.1208

Time taken to build model: 0 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===

Correlation coefficient                  0.9945
Mean absolute error                   4053.821
Root mean squared error               4578.4125
Relative absolute error                 13.1339 %
Root relative squared error             10.51   %
Total Number of Instances                7
```

Resultados da previsão:

```
SellingPrice = (-26.6882    * 3198) +
   (7.0551      * 9669) +
   (43166.0767 * 5) +
   (42292.0901 * 1)
   - 21661.1208

 SellingPrice = 219,328
```

No projeto EDA2 do PyCharm, criar um ficheiro chamado decisiontree_regression_House.py para guardar o código para as seguintes instruções:

1. Ler os ficheiros House.csv e House_new que se encontram no Moodle:

```
import pandas as pd  # To read data
data = pd.read_csv('caminho_fich/House.csv')
data_new = pd.read_csv('caminho_fich/House_new.csv')
```

2. Definir quais são as variáveis independentes (X) e dependente (y)

```
X = data[['HouseSize', 'LotSize', 'Bedrooms', 'Granite', 'Bathroom']]
Y = data[['SellingPrice']]
```

3. Criar e fazer fit da árvore de decisão

```
from sklearn.tree import DecisionTreeRegressor
regr_1 = DecisionTreeRegressor()
regr_1.fit(X, Y)
```

4. Ver o modelo criado

```
from sklearn.tree import export_text
r = export_text(regr_1, feature_names=['HouseSize', 'LotSize', 'Bedrooms', 'Granite',
'Bathroom'])
print(r)
```

Resultado

```
|--- LotSize <= 17075.00
|   |--- HouseSize <= 2690.00
|   |   |--- Bathroom <= 0.50
|   |   |   |--- value: [189900.00]
|   |   |--- Bathroom >  0.50
|   |   |   |--- value: [195000.00]
|   |--- HouseSize >  2690.00
|   |   |--- HouseSize <= 3388.00
|   |   |   |--- HouseSize <= 3115.00
|   |   |   |   |--- value: [230000.00]
|   |   |   |--- HouseSize >  3115.00
|   |   |   |   |--- value: [224900.00]
|   |   |--- HouseSize >  3388.00
|   |   |   |--- Bedrooms <= 5.50
|   |   |   |   |--- value: [197900.00]
|   |   |   |--- Bedrooms >  5.50
|   |   |   |   |--- value: [205000.00]
|--- LotSize >  17075.00
|   |--- value: [325000.00]
```

5. Desenhar a árvore de decisão

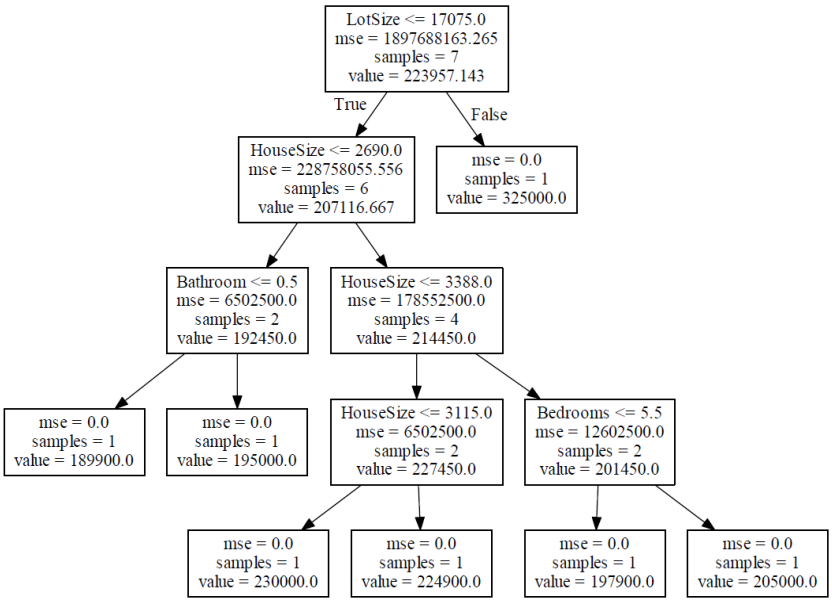
```
from sklearn import tree
# Create DOT data
dot_data = tree.export_graphviz(regr_1, out_file=None,
                                feature_names=['HouseSize',    'LotSize',    'Bedrooms',
'Granite', 'Bathroom'],
                                class_names=regr_1.classes_)

# Draw graph
import pydotplus
graph = pydotplus.graph_from_dot_data(dot_data)

# Show graph
from IPython.display import Image

Image(graph.create_png())
graph.write_pdf('dt_regr.pdf')
```

6. Verificar o resultado guardado no ficheiro dt_regression.pdf dentro do projeto EDA2



7. Avaliar o modelo

```
from sklearn.metrics import mean_squared_error
predictions = regr_1.predict(X=data[['HouseSize', 'LotSize', 'Bedrooms', 'Granite',
                      'Bathroom']])
mse = mean_squared_error(data[['SellingPrice']], predictions)
print("MSE:", mse)
```

Resultado:

```
MSE: 0.0
```

Nota: O resultado dá 0 de erro, porque o modelo está *overfitted* ao conjunto de treino

8. Aplicar o modelo aos dados novos

```
predictions = regr_1.predict(X=data_final_new[['HouseSize', 'LotSize', 'Bedrooms',
                                               'Granite', 'Bathroom']])
print("Predicted price: % d\n"% predictions)
```

9. Resultado

```
Predicted price:  224900
```

Regressão Linear

Regressão Linear Simples

Processo "Manual"

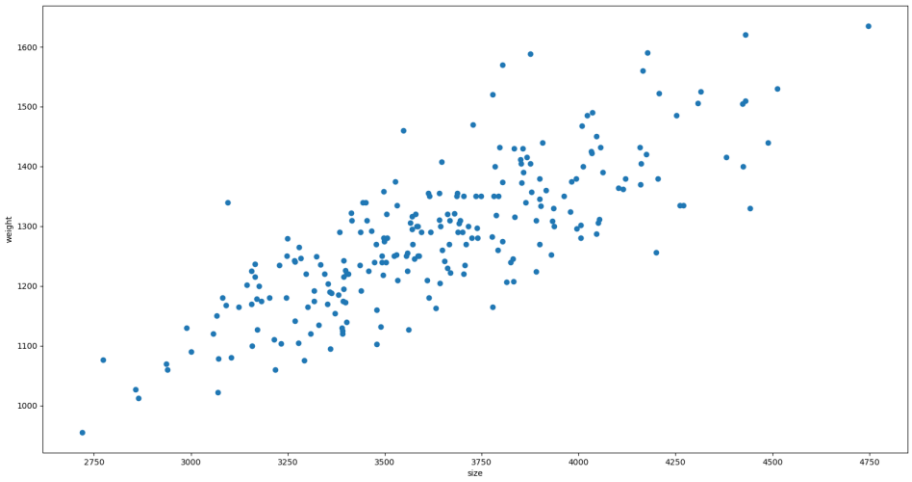
1. Ler o ficheiro sizeweight.csv (que se encontra no Moodle)

```
import pandas as pd
data = pd.read_csv('caminho_fich/sizeweight.csv')
```

2. Desenhar o gráfico dos dados

```
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (20.0, 10.0)
plt.scatter('size', 'weight', data=data)
plt.xlabel('size')
plt.ylabel('weight')
plt.show()
```

Resultado:



3. Definir as variáveis independente (X) e dependente (Y)

```
X = data['size'].values
Y = data['weight'].values
```

4. Calcular β_0 e β_1 e mostrar a equação da reta

```
import numpy as np

mean_x = np.mean(X)
mean_y = np.mean(Y)
n = len(X)

numer = 0
denom = 0
for i in range(n):
    numer += (X[i] - mean_x) * (Y[i] - mean_y)
    denom += (X[i] - mean_x) ** 2
b1 = numer / denom
b0 = mean_y - (b1 * mean_x)

print('y =',b1,'* x +', b0)
```

Resultado:

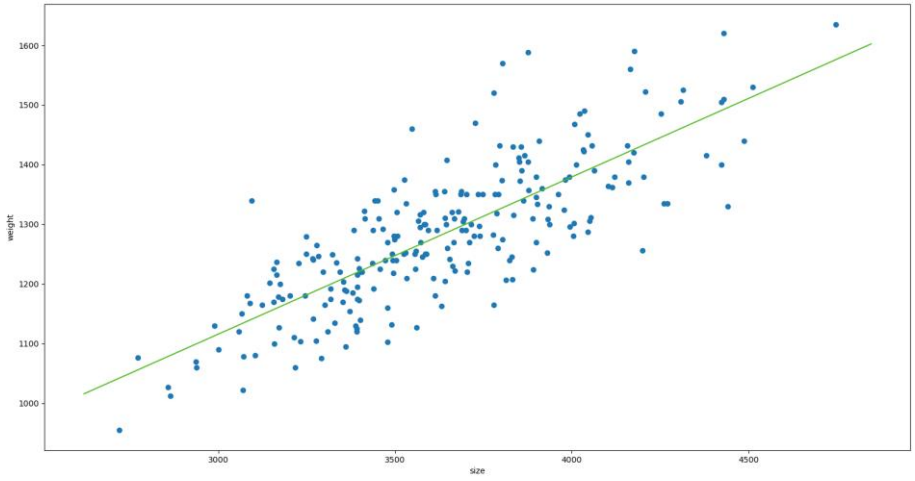
```
y = 0.26342933948939945 * x + 325.57342104944223
```

5. Desenhar o gráfico dos dados e a reta obtida

```
max_x = np.max(X) + 100
min_x = np.min(X) - 100
x = np.linspace(min_x, max_x, 1000)
y = b0 + b1 * x

plt.rcParams['figure.figsize'] = (20.0, 10.0)
plt.scatter('size', 'weight', data=data)
plt.xlabel('size')
plt.ylabel('weight')
plt.plot(x, y, color='#52b920')
plt.show()
```

Resultado:



6. Avaliar o modelo: calcular o R_2

```
ss_t = 0
ss_r = 0
for i in range(X.size):
    y_pred = b0 + b1 * X[i]
    ss_t += (Y[i] - mean_y) ** 2
    ss_r += (Y[i] - y_pred) ** 2
r2 = 1 - (ss_r/ss_t)
print("r2 (manual) =",r2)
```

Resultado:

```
r2 (manual) = 0.6393117199570003
```

Utilizando "scikit learn"

7. Importar a biblioteca

```
from sklearn.linear_model import LinearRegression
```

8. Reformatar a variável independente

```
X = X.reshape((X.size, 1))
```

9. Criar o modelo

```
reg = LinearRegression()
```

10. Fazer fit ao modelo

```
reg = reg.fit(X, Y)
```

11. Calcular as previsões do modelo

```
Y_pred = reg.predict(X)
```

12. Calcular e mostrar o R^2

```
r2_score = reg.score(X, Y)
print("r2 (LinearRegression) =",r2_score)
```

Resultado:

```
r2 (LinearRegression) = 0.639311719957
```

Regressão Linear Múltipla

Replicação do exemplo com o Weka

O objetivo deste exercício é reproduzir o "Laboratório 1 – Data Mining: Regressão" resolvido com o Weka. Num primeiro momento, deve relembrar os resultados desse exercício.

Resultados do modelo:

```
Linear Regression Model

SellingPrice =

    -26.6882 * HouseSize +
      7.0551 * LotSize +
  43166.0767 * Bedrooms +
  42292.0901 * Bathroom +
 -21661.1208

Time taken to build model: 0 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===

Correlation coefficient                  0.9945
Mean absolute error                   4053.821
Root mean squared error               4578.4125
Relative absolute error                 13.1339 %
Root relative squared error             10.51   %
Total Number of Instances                7
```

Resultados da previsão:

```
SellingPrice = (-26.6882    * 3198) +
   (7.0551      * 9669) +
   (43166.0767 * 5) +
   (42292.0901 * 1)
   - 21661.1208

 SellingPrice = 219,328
```

No projeto EDA2 do PyCharm, criar um ficheiro chamado linearRegression_House.py para guardar o código para as seguintes instruções:

13. Ler os ficheiros House.csv e House_Final_new.csv (que se encontram no Moodle)

```
import pandas as pd
data = pd.read_csv('caminho_fich/House.csv')
data_final_new = pd.read_csv('caminho_fic/House_final_new.csv')
```

14. Criar um modelo[1] de regressão linear para o dataset original.

```
from sklearn.linear_model import LinearRegression
linear_regressor = LinearRegression()
linear_regressor.fit(data[['HouseSize', 'LotSize', 'Bedrooms', 'Granite', 'Bathroom']],
data['SellingPrice'])
```

A criação do modelo de regressão linear é feita com recurso à criação de um LinearRegressor, seguido da execução do fit do modelo. Para isso, é necessário importar a biblioteca LinearRegression a partir de sklearn.linear_model. A documentação importante para este passo pode ser encontra neste link. Pretende-se que o modelo considere as variáveis independentes HouseSize, LotSize, Bedrooms, Granite, Bathroom e a variável dependente SellingPrice.

15. Visualizar os coeficientes e o intercept do modelo.

1

```
print("coefs:", linear_regressor.coef_, sep=" ")
print("intercept:", linear_regressor.intercept_, sep=" ")
```

O resultado obtido deve ser o seguinte

```
coefs: [-2.69307835e+01 6.33452410e+00 4.42937606e+04 7.14067629e+03  4.31791999e+04]
intercept: -21739.29666506665
```

Os coeficientes encontram-se em notação científica. Para evitar isto, acrescentamos antes dos prints as seguintes linhas:

```
import numpy as np
np.set_printoptions(formatter={'float_kind': '{:f}'.format})
```

Desta forma, o resultado obtido é o seguinte:

```
coefs: [-26.930784 6.334524 44293.760584 7140.676293 43179.199889]
intercept: -21739.29666506665
```

Os coeficientes são apresentados pela ordem dos atributos. Isto significa que a equação de regressão é a seguinte:

```
SellingPrice = -26.930784 * HouseSize +
                   6.334524 * LotSize +
               44293.760584 * Bedrooms +
                7140.676293 * Granite +
               43179.199889 * Bathroom +
               -21739.29666506665
```

Como Podemos verificar, este modelo (ao contrário do modelo gerado pelo Weka), considera o atributo "Granite". Para deixar de considerar este atributo, temos de o eliminar da função fit do ponto 2:

```
from sklearn.linear_model import LinearRegression
linear_regressor = LinearRegression()
linear_regressor.fit(data[['HouseSize',     'LotSize',     'Bedrooms',     'Bathroom']],
data['SellingPrice'])
```

Ao voltar a executar o código, temos o seguinte output:

```
coefs: [-26.688240 7.055124 43166.076944 42292.090237]
intercept: -21661.12129661304
```

O que quer dizer que foi obtida a seguinte equação de regressão:

```
SellingPrice = -26.688240 * HouseSize +
                   7.055124 * LotSize +
               43166.076944 * Bedrooms +
               42292.090237 * Bathroom +
               -21661.12129661304
```

Que corresponde, aproximadamente, à equação obtida no Weka

16. Obter algumas métricas de performance do modelo

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
import math
predictions = linear_regressor.predict(data[['HouseSize', 'LotSize', 'Bedrooms',
'Bathroom']])
print("\n= METRICS =")
print('mean_absolute_error: ', mean_absolute_error(data['SellingPrice'], predictions),
sep="")
print('Root mean_squared_error: ', math.sqrt(mean_squared_error(data['SellingPrice'],
predictions)), sep="")
```

As métricas a calcular são o Mean Absolute Error e o Root Mean Squared Error. Para isso, é necessário importar as funções mean_absolute_error e mean_squared_error da biblioteca sklearn.metrics, cuja documentação pode ser encontrada neste link. Como a função mean_squared_error calcula o Mean Squared Error, é preciso aplicar-lhe a raiz quadrada, utilizando a função sqrt da biblioteca math. O output será:

```
= METRICS =
mean_absolute_error: 4053.8210607484966
Root mean_squared_error: 4578.412476004145
```

Estes valores são semelhantes aos obtidos com o Weka

17. Calcular e visualizar as previsões do modelo para os dados novos.

```
predictions2 = linear_regressor.predict(data_final_new[['HouseSize', 'LotSize',
'Bedrooms', 'Bathroom']])
print("\nmodel final predictions:", predictions2, sep=" ")
```

As previsões são calculadas recorrendo à função predict. O output é o seguinte:

```
model final predictions: [219328.357193]
```

Este resultado é semelhante ao obtido com o Weka.