FE07

Universidade do Minho
Escola de Engenharia

Curso: Mestrado em Engenharia Biomédica
U.C.: Aprendizagem e Extração do Conhecimento

Ficha de Exercícios 07	
Docente:	Larissa Montenegro | Hugo Peixoto | António Abelha
Tema:	Introdução a Python & a Machine Learning Algorithms
Ano Letivo:	2022-2023 – 1º Semestre
Duração da aula:	2 horas

1. Introdução Python

introdução à linguagem de programação Python e ao ambiente utilizado para os exercícios do curso: Colab. Colab é um ambiente de desenvolvimento Python que corre no navegador utilizando o Google Cloud.

Please go to: https://colab.research.google.com
→ New notebook
→ Change Name Untitled1.ipynb to IntroToPython.ipynb

Libraries

Python libraries for mathematics, science and engineering. It is an add-on to Python that you will need for machine learning:

- **NumPy:** A foundation for SciPy that allows you to efficiently work with data in arrays. You will prepare your data as NumPy arrays for modeling in machine learning algorithms.
- **Matplotlib:** Allows you to create 2D charts and plots from data. You will use Matplotlib to create plots and charts of your data.
- **Pandas:** Tools and data structures to organize and analyze your data. You will use Pandas to load explore and better understand your data.
- **scikit-learn** The scikit-learn library is how you can develop and practice machine learning in Python.

run the following Python code to print the versions of the installed Python version and libraries.

```
python -version

import matplotlib
matplotlib.__version__

import pandas
pandas.__version__

import numpy
numpy.__version__
```

2. Language syntax to be able to read and understand Python code

2.1 Strings

```
# Strings
data = 'hello world'
print(data[0])
print(len(data))
```

2.2 Numbers

```
# Numbers
value = 123.1
print(value)
value = 10
```

```
print(data)

# Results
h
11
hello world
```

```
print(value)

# Results
123.1
10
```

2.3 Boolean

```
# Boolean
a = True
b = False
print(a, b)

# Results
(True, False)
```

2.4 Multiple Assignment

```
# Multiple Assignment
a, b, c = 1, 2, 3
print(a, b, c)

# Results
(1, 2, 3)
```

2.5 No Value

```
# No value
a = None
print(a)

# Results
None
```

3. Flow Control

There are three main types of flow control that you need to learn: If-Then-Else conditions, For-Loops and While-Loops.

3.1 If-Then-Else Conditional

```
value = 99
if value == 99:
   print 'That is fast'
elif value > 200:
    print 'That is too fast'
else:
    print 'That is safe'
```

3.2 For-Loop

```
# For-Loop

for i in range(10):
  print i
```

3.3 While-Loop

```
# While-Loop

i=0
while i < 10:

print i i += 1
```

4. Data Structures

There are three data structures in Python that you will find the most used and useful. They are tuples, lists and dictionaries.

4.1 Tuple: Tuples are read-only collections of items.

```
a = (1, 2, 3)
print a
```

4.2 List: Lists use the square bracket notation and can be index using array notation.

```
mylist = [1, 2, 3]

mylist.append(4)

for value in mylist:
  print value
```

4.3 Dictionary: Dictionaries are mappings of names to values, like key-value pairs. Note the use of the curly bracket and colon notations when defining the dictionary.

```
mydict = {'a': 1, 'b': 2, 'c': 3}
print('A value is: ', mydict['a'])
mydict['a'] = 11
print('New A value is: ', mydict['a'])
print("Keys:", mydict.keys())
print("Values:", mydict.values())
for key in mydict.keys():
  print(mydict[key])
```

4.4 Functions

```
# Sum function
def mysum(x, y):
  return x + y

# Test sum function
result = mysum(1, 3)
print(result)
```

5. NumPy

Numpy provides the foundation data structures and operations. These are arrays (ndarrays) that are efficient to define and manipulate.

5.1 Create Array

```
# define an array
import numpy

mylist = [1, 2, 3]
myarray = numpy.array(mylist)
print(myarray)
print(myarray.shape)
```

5.2 Access Data: Array notation and ranges can be used to efficiently access data in a NumPy array.

```
# Access values
import numpy
mylist = [[1, 2, 3], [3, 4, 5]]
myarray = numpy.array(mylist)
print(myarray)
print(myarray.shape)
print("First row: %s") % myarray[0]
print("Last row: %s") % myarray[-1]
print("Specific row and col: %s") % myarray[0, 2]
```

```
print("Whole col: %s") % myarray[:, 2]
```

6. Matplotlib

Matplotlib can be used for creating plots and charts

```
# basic line plot
import matplotlib.pyplot as plt import numpy

myarray = numpy.array([1, 2, 3])
plt.plot(myarray)
plt.xlabel('some x axis')
plt.ylabel('some y axis')
plt.show()
```

7. Introduction to Linear Regression

Ordinary least squares Linear Regression.
Linear regression fits a linear model with coefficients w = (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset and the targets predicted by the linear approximation.

Methods

Function	Description
fit(x, y[, sample_weight])	fit linear model.
get_params([deep])	get parameters for this estimator.
predict(x)	predict using the linear model.
score(x, y[, sample_weight])	return the coefficient of determination of the prediction.
set_params(**params)	set the parameters of this estimator.

7.1 Implementation Linear Regression

From the implementation point of view, this is just plain Ordinary Least Squares
Implement the linear regression function by importing the following library f*rom sklearn.linear_model import LinearRegression* and *Numpy*

Create a variable X of an array of an array, such as:

```
X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
```

Create a varible y = 1 * x_0 + 2 * x_1 + 3

```
y = np.dot(X, np.array([1, 2])) + 3

# Fit linear model.
reg = LinearRegression().fit(X, y)

# Return the coefficient of determination of the prediction. The best possible score is
1.0
reg.score(X, y)

# coef_ gives you an array of weights estimated by linear regression. It is of shape
(n_targets, n_features)
reg.coef_

# independent term in the linear model
reg.intercept_

# Predict using the linear model.
reg.predict(np.array([[3, 5]]))
```

Exercise1: Predicting the diabetes disease progression

The Data Set used in this exercise is about diabetes available in the file scikit datasets repository. This dataset contains data from diabetic patients and contains certain features such as their bmi, age, blood pressure and glucose levels which are useful in predicting the diabetes disease progression in patients.

Data Understanding

The diabetes dataset is loaded using load_diabetes() function:

```
diabetes = datasets.load_diabetes(as_frame=True)
```

You can check the type of variable and show the dataset's keys by:

```
print(list(diabetes))
```

You can check the content of each key by

```
print(type(diabetes['key']))
```

Task 1: Which key give you the description of the dataset? How many instances does the dataset has? How many attributes and which ones? Which Column give the quantitative measure of disease progression?

Task 2: Plot each feature data individually against the target data (Fig1). Do not plot the sex attribute. The plot should be with the dispersion *scatter* function. Add title, Axis names to the figures. How could all plots figures be added as subplots?

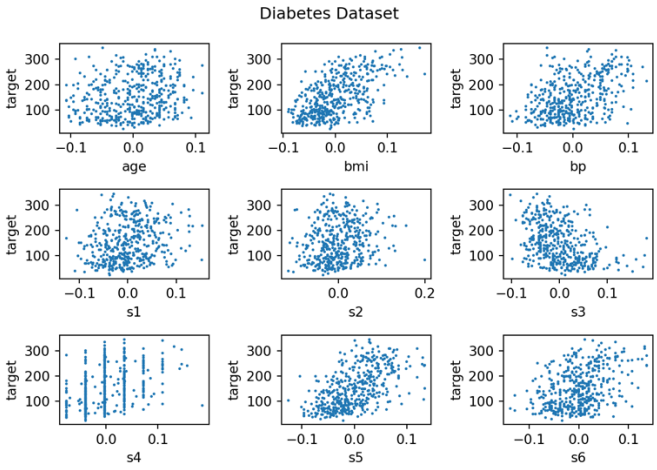

Figure 1 - feature data individually against the target data

Task 3: based on the graph, which attribute would look like more associated to the diabetes progression?

Data Processing

In this section, we will prepare the dataset for modelling.

Task4. For the prediction task, you should choose just one attribute (feature) based on your response on task 3 in data understanding. Remember to load the column of the targets as well in a variable name *diabetes_y.*
To do so:

```
# Get all Features names
FeaturesName = diabetes['feature_names']
print(FeaturesName)
```

```
# Load the Chosen column, for example for column sex
diabetes_X = diabetes['data']['sex']
print(diabetes_X)
```

Task 5. Split the feature and target data into training (90%) /test sets (10%). How many samples has the training set and how many samples has the test set?

Modelling

Task 6. Create linear regression object
Task 7. Train the model using the training sets
Task 8. Make predictions using the testing set

Evaluation

Task 9. Calculate the coeficientes
Task 10. Calculate the mean_squared_error(y_test,y_predicted)
Task 11. Calculate the Coefficient of determination r2_score(y_test,y_predicted)

Task 12. Plot outputs: scatter for diabetes_X_test and diabetes_y_test and plot for diabetes_X_test and diabetes_y_pred

The straight line can be seen in the plot, showing how linear regression attempts to draw a straight line that will best minimize the residual sum of squares between the observed responses in the dataset, and the responses predicted by the linear approximation.

Based on the feature selected how did the diabetes prediction? Repeat the task choosing another feature. Compare your results and what can be concluded?

Exercise2: Medical Costs Prediction

Build a simple Linear Regression model to predict medical costs. You will analyze a medical cost dataset and use a regression model. The Dataset can be downloaded from the following link:
https://www.kaggle.com/mirichoi0218/insurance or via WeTransfer https://we.tl/t-30pWv2PmCg

Steps to follow:

- Load the data,
- Analyze information and graphically represent numerical categories,
- Convert object categories to label (dummy) variables,
- fit Linear Regression models,
- Predict values using the fitted model, and
- Measure the accuracy of the model.

For this task, you will use the following libraries numpy, panda, seaborn and matplotlib.pyplot

Pandas DataFrame is a **two-dimensional size-mutable, potentially heterogeneous tabular data structure** with labeled axes (rows and columns). It is built on top of another package named Numpy, which provides support for multi-dimensional array.

Seaborn is **a Python data visualization library based on matplotlib**. It provides a high-level interface for drawing attractive and informative statistical graphics.

1. Load the dataset and Analyze the Dataset

To import the csv data as a DataFrame in Python use the following function :

```
insurance_df = pandas.read_csv('insurance.csv ')
```

Tasks:

- How many and Which categories are included in our data set? Use function variableName.columns
- Print the first five rows of the DataFrame to see how our data look in the table. Use function insurance_df.head()
- The Number of entries for each category and type of data is presented. Use function insurance_df.info()
- Characteristics of all numerical categories. Use the function insurance_df.describe()
- Graphically represent the numerical data. Use function insurance_df.pairplot(df). From the Graph, what can you conclude from the correlation between the categories?

Pairplot is the easiest way to see all mutual correlations between different categories and distributions of data for each category separately, for all numerical categories

2. Convert object categories to label (dummy) variables,

Before we go into the prediction, we need to prepare the dataset. There are three columns that the values are categorical variables: Region, sex, and smokers. To check the number of samples inside the category, use the function:

```
insurance_df['region'].value_counts()
```

We could determine the number of samples for each category, as we did with the “region”.

One Hot Encoding

We will use one hot encoding to present all these categories in a way suitable for processing. One hot encoding is the procedure of transforming categorical variables into binary vectors.

Category Regions:

```
[southeast,southwest,northwest,northeast]
```

One hot encoding will transform affiliation to a specific region to the vector of four elements, for example:

```
[0,0,1,0] - Third element is 1, so this individual is from the northwest.
[1,0,0,0] - First Element is 1, So this individual is from southwest.
```

Task: Transform categorical variables sex, smoker, and region using One hot encoding.

Example:

```
region_dummy = pd.get_dummies(df['region'])
```

The obtained categories need to be add to the actual data frame and the originals need to be removed. At the end, we Will have now seven categories, as the smokers will be as well divided into two new categories: non-smoker and nicotian.

```
df = pd.concat([df,sex_dummy,smoker_dummy,
                    region_dummy], axis=1)

df.rename(columns={'no': 'non-smoker',
              'yes': 'nicotian'}, inplace=True)

df = df.drop(['sex','smoker','region'], axis=1)

df.head(10)
```

3. Training and Test set

We have prepared our data for processing. Now, it needs to be split into training and test set. To do so, we will use the function train_test_split

```
from sklearn.model_selection import train_test_split
```

Eleven data-frame categories will be used as inputs of the model. We want to fit our model according to the "charges" category so that output variable Y will be "charges".

Task: Define the variable input X with the eleven categories and the output y with charges. Following, split the dataset into X_train, X_test, y_train, y_test using the function train_test_split. The test size is 40% of the dataset.

4. Fit Linear Regression models

Tasks:

- Proceed with fitting the Linear Regression model.
- Test the trained model for predicting new "charges" values (by using prepared X_test data which is obtained by *train_test_split* function) and print the predictions
- Present a graphic (scatter) comparison of the expected values (y_test) and predicted values (predictions)
- Graph error distribution graph of our predictions with

```
sns.distplot((y_test-predictions), bins=50)
```

- calculate and print mean absolute error (MAE) and mean squared error (MSE) for the predictions. These measures will represent quality parameters of our model (achieved prediction accuracy).

Use:

```
from sklearn import metrics
metrics.mean_absolute_error()
metrics.mean_squared_error()
```