Introduction to Scikit-Learn : A Python Machine Learning Library

Requirements for the work: the exercise 6 should be uploaded on the DVO repository before the deadline. You should create your colab file correctly named.

1 Introduction

When you're working on a machine learning project, the most tedious steps are often data cleaning and preprocessing. Especially when you're working in a Jupyter Notebook, running code in many cells can be confusing.

The Scikit-learn library has tools called Pipeline and ColumnTransformer that can really make your work easier. Instead of transforming the dataframe step by step, the pipeline combines all transformation steps. You can get the same result with less code. It's also easier to understand data workflows and modify them for other projects. First let we introduce briefly the scikit-learn library.

1.1 What is scikit-learn?

Scikit-learn provides a range of supervised and unsupervised learning algorithms through a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.

The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn. This stack includes:

- NumPy: Base n-dimensional array package
- SciPy: Fundamental library for scientific computing
- Matplotlib: Comprehensive 2D/3D plotting
- IPython: Enhanced interactive console
- Sympy: Symbolic mathematics
- Pandas: Data structures and analysis

1.2 What is the Scikit-learn Pipeline?

Before training a model, you split your data into two datasets: a training set and a test set. Each dataset will go through the data cleaning and preprocessing steps before you inject it into a machine learning model.

Pipelines are very common in ML-systems since there is a lot of data to manipulate and many data transformations to apply. It's not efficient to write repetitive code for the training set and the test set. This is when the scikit-learn pipeline comes into play.

Scikit-learn pipeline is a powerful tool to standardise your operations and chain them in a sequence, make unions and finetune parameters. For example as a typical data science workflow would have these steps:

1. get the training data
2. clean, preprocess, transform data
3. train a ML-model
4. evaluate and optimize the model
5. clean, preprocess and transform new data
6. fit the model on new data to make prediction

You may remark that data preprocessing has to be done at least two times in the workflow. As painful and time-consuming as this step is, how useful it would be if only we could automate this process and apply it to any future new datasets. So sklearn pipeline systemises easily the process and therefore makes it reproducible.

1.3 What is the Scikit-learn ColumnTransformer?

ColumnTransformer will transform each group of dataframe columns separately and combine them later. This is useful in the data preprocessing process. Indeed, in a structured data table, we often have columns with qualitative variables(categorical attributes) and others with quantitative variables (numerical attributes). We may also encounter cases where We want to vary the types of pre-processing according to the columns like standardization of certain columns, centering for others...Etc. To perform all these actions in a single object, We have to use the columnTransformer class of scikit-learn able to handle easily data variety.

Quick start with GoogleColab Environment

Colab (or "Colaboratory") allows you to write and execute Python code in your browser with 1)No configuration requirement; 2)Free access to GPUs; 3)Easy sharing the code.

Step1: Create your own colab

Open a new colab in GoogleColab for Python3 by starting a new environment from . You can name your file as your $firstName\_lastName\_TP01.ipynb$. This will save you from installing Python on your computer or managing libraries as to switch to Pandas X.0 or install the last Scikit-Learn release !.

Once connected to Google Colab, just create a new Python notebook:

Step2: Import APIs

Two concurrent high level API exist allowing us to built, train and test different models based on scikit-learn. Let discover these APIs by doing. The first step that we need is ***Working with data***, let we download the dataset with Pandas:

scikit-learn APIs

scikit-learn provides modules and classes as **Pipeline, ColumnTransformer**

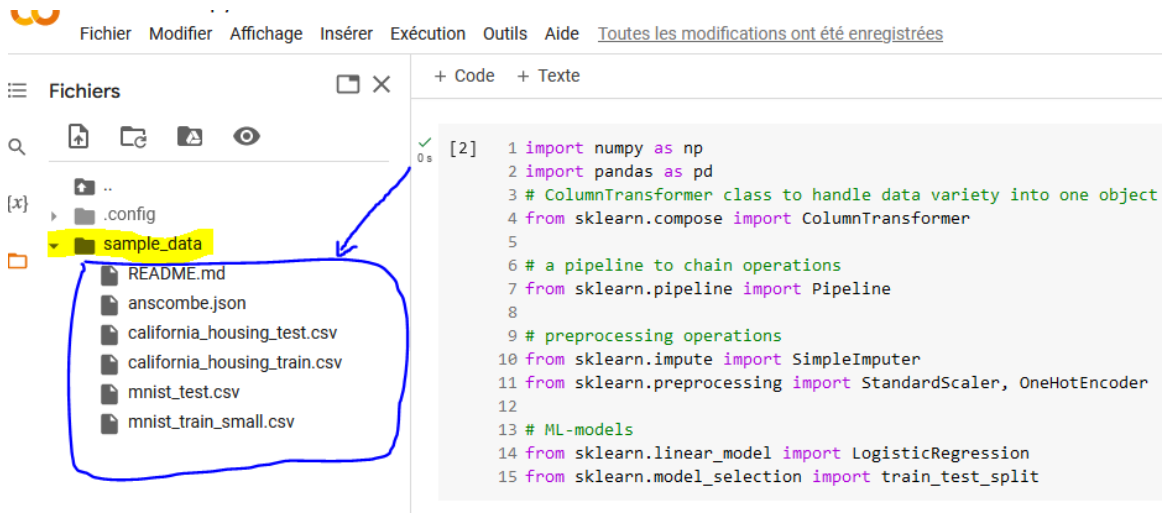
```
import numpy as np
import pandas as pd
# ColumnTransformer class to handle data variety into one object
from sklearn.compose import ColumnTransformer

# a pipeline to chain operations
from sklearn.pipeline import Pipeline

# preprocessing operations
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder

# ML-models
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

When the code above is running successfully, a sample_Data repository is added to your colab path. You can check its content by clicking on the ”files” at the left:



Exercise 1: download the dataset

To start with a simple scikit-learn example, you may download a simple dataset to your local machine from this link: WineQuality The two datasets are related to red and white variants of the Portuguese ”Vinho Verde” wine. For more details, consult: www.vinhoverde.pt or the reference [Cortez et al., 2009]. Due to privacy and logistic

issues, only physicochemical (inputs) and sensory (the output) variables are available (e.g. there is no data about grape types, wine brand, wine selling price, etc.).

These datasets can be viewed as classification or regression tasks. The classes are ordered and not balanced (e.g. there are many more normal wines than excellent or poor ones). Outlier detection algorithms could be used to detect the few excellent or poor wines. Also, we are not sure if all input variables are relevant. So it could be interesting to test feature selection methods. Now try to download the dataset and to print the 3 lines at the head. The object containg the hole dataset is "wine":

```
#You can load the data from from the UCI ML website:
#df = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/
    winequality-red.csv',sep=';')
#or after downloading in your local repository
wine = pd.read_csv("/content/sample_data/winequality-red.csv", sep=';')
print(wine.head(3))
```

Exercise 2: Get data information

You can check for missing data by adding the null elements:

```
print winedf.isnull().sum()
```

The result of the print is 0 for each attribute. Here you can find 12 attributes or commonly named features if you call the function info() which is useful method to get a quick description of the data, in particular the total number of rows, each attribute's type and the number of nonnull values:

```
print(wine.info())
```

We have 11 float features (float64) and 1 integer feature (int64). You can display now the histogramm of each feature using pyplot class. The function hist() allows you to evaluate the histogramm of any numerical vector:

```
import matplotlib.pyplot as plt
wine.hist(bins=50,figsize=(20,15))
plt.show()
```

In many cases, we need to show a quick summary of the data and hence to display numerical features summary. To do this, we call the method **describe()**

```
wine.describe()
plt.show()
```

Exercise 3: Basic statistical analysis and visualization

The count, mean, min, max rows are sel-explanatory. The std row shows the standard deviation which measure how dispersed the values are. The 25%, 50% and 75% rows show the corresponding percentiles: a percentile indicates the value below which a given percentage of observations in a group of observations fall. For example 25% of wines (first quartile) has a fixed acidity lower than 7.

Boxplots summarize the distribution of each attribute, drawing a line for the median (middle value) and a box around the 25th and 75th percentiles (the middle 50% of the data). The whiskers give an idea of the spread of the data and dots outside of the whiskers show candidate outlier values (values that are 1.5 times greater than the size of spread of the middle 50% of the data).

```
wine.plot(kind='box', subplots=True, layout=(3,4), sharex=False, sharey=False)
plt.show()
```

Exercise 4: Correlation Matrix Plot

Correlation gives an indication of how related the changes are between two variables. If two variables change in the same direction they are positively correlated. If the change in opposite directions together (one goes up, one goes down), then they are negatively correlated.

You can calculate the correlation between each pair of attributes. This is called a correlation matrix. You can then plot the correlation matrix and get an idea of which variables have a high correlation with each other.

This is useful to know, because some machine learning algorithms like linear and logistic regression can have poor performance if there are highly correlated input variables in your data.

```
import numpy
correlations = wine.corr()
# plot correlation matrix
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = numpy.arange(0,12,1)
names= wine.head(0)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
#ax.set_xticklabels(names)
ax.set_yticklabels(names)
plt.show()
```

A scatterplot shows the relationship between two variables as dots in two dimensions, one axis for each attribute. You can create a scatterplot for each pair of attributes in your data. Drawing all these scatterplots together is called a scatterplot matrix.

Scatter plots are useful for spotting structured relationships between variables, like whether you could summarize the relationship between two variables with a line. Attributes with structured relationships may also be correlated and good candidates for removal from your dataset.

```
from pandas.plotting import scatter_matrix
scatter_matrix(wine)
plt.show()
```

Exercice 5: SKlearn Pipeline

Machine Learning pipeline, theoretically, represents different steps including data transformation feature scaling, feature selection / extraction and training/testing the models. Python Sklearn Pipeline (sklearn.pipeline) package will help automate these ML activities instead of going through the model fitting and data transformation steps for the training and test datasets separately. Three steps are required to use a pipeline:

1. pipeline creation or instantiating

2. pipeline fitting
3. pipeline prediction

Pipeline creation

To create the pipeline we have to thing the list of successive tasks. For example, let us define a simple pipeline containing a data preprocessing task like ***StandardScaler*** operator, ***PCA*** for data reduction, and ***RandomForestClassifier*** as a model classifier. ***StandardScaler*** is a class from sklearn.preprocessing package and ***PCA*** is a class from sklearn.decomposition package. Here the code to create a simple pipeline named ***ML_pipeline***

```
#import classes
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import make_pipeline

#create the pipeline
ML_pipeline = make_pipeline(StandardScaler(),
                        PCA(n_components=2),
                        RandomForestClassifier(criterion='entropy', n_estimators=10,
   max_depth=4, random_state=1))
```

Let we learn about these three functions: ***StandardScaler()***, ***PCA***, and ***RandomForestClassifier***.

StandardScaler()

Standardize features by removing the mean and scaling to unit variance.

sklearn.preprocessing.StandardScaler!

```
class sklearn.preprocessing.StandardScaler(*, copy=True, with_mean=True, with_std=
   True)
```

The standard score of a sample x is calculated as:

$X = (x - u)/s$ where u is the mean of the training samples or ***zero*** if ***with_mean=False***, and s is the standard deviation of the training samples or ***one*** if ***with_std=False***.

A Reminder: Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual features do not more or less look like standard normally distributed data (e.g. Gaussian with 0 mean and unit variance). Other methods could be used to normalize samples. We will see this topic in the next WP.

Pipeline fitting

Now our pipeline is ready to fit the list of tasks on the data. As our purpose here is to classify the wine into 9 categories from 0 to 8 according to the column ***quality***, we have to decide about the output feature and the explanatory features:

- ***quality***: is the output feature or the class target. It has 9 labels which range from 0 to 8

- ***'fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'sulfur dioxide ratio', 'density', 'pH', 'sulphates', 'alcohol'***: are explanatory features

Let we note x the matrix of explanatory feature's values and y the vector of label's values. We use the function ***loc*** to select these data:

```
x=wine.loc[:, 'fixed acidity':'alcohol']
y=wine.loc[:, 'quality']
```

A Reminder: $loc[]$ Access a group of rows and columns by label(s) or a boolean array.

Training and test datasets

Sklearn provides a few functions to split datasets into multiple subsets in various ways. The simplest function is ***train_test_split*** which is the pretty much the same thing as the function ***split_train_test*** with a couple of additional features. First, there is a random_state parameter allowing to set the random generator seed. Second you can pass it multiple datasets with the same number of rows and it will split them on the same indices:

sklearn.model_selection.train_test_split!

```
sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None,
    random_state=None, shuffle=True, stratify=None)
```

Split arrays or matrices into a random train and test subsets. Try this simple example :

```
import numpy as np
from sklearn.model_selection import train_test_split
X, Y = np.arange(0,12,1,dtype=int).reshape((6, 2)), range(6)
print("X: ",X)
print("Y: ",list(Y))
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.33,
    random_state=42)
print("x_train: ",x_train)
print("x_test: ",X_test)
print("y_train: ",y_train)
print("y_test: ",y_test)
```

fitting

When the pipeline's fit() method is called, the fit_transform() is sequentially called on all transformers passing the output of each call as the parameter to the next call until it reaches the final estimator for which it calls the fit() method.

```
ML_pipeline.fit(X_train, y_train)
```

The estimator used in this dataset is Random Forest Classifier. A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree (see Figure 1). We will discover other estimators later through this module.

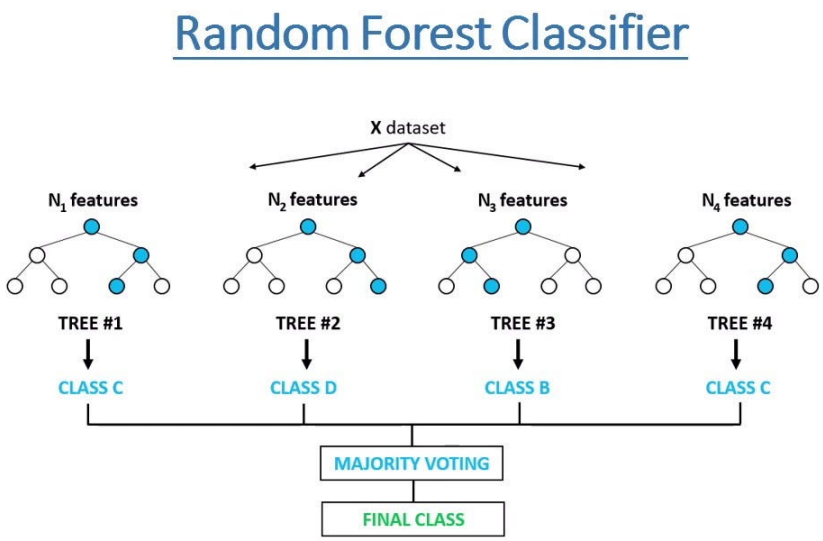


Figure 1: Random Forest Classifier(Ankit Chauhan)

New data prediction

Now is the time to evaluate the trained model on new data never used in the training. However we are interesting to measure to accuracy of the model to predict the quality value of any new wine sample (test dataset) when only the features of each new sample is used used to estimate the outpout value of the quality feature (the class prediction). There is nothing special about this process, just get the predictors and the labels from the test set. When the function score() is called, the test data is transformed, and a score is calculated with the final estimator.

```
y_pred = ML_pipeline.predict(X_test)
test_acc = ML_pipeline.score(X_test, y_test)
print(f'Test accuracy: {test_acc:.3f}')
print(y_test-y_pred)
```

Model Storage

If we need to use the same model on other datasets another time, we can serialize the fitted model in a simple file with an extension pkl. Developed by Python, the PKL extension refers to one of its modules, which helps to serialize (***pickling***) and de-serializing (***unpickling***) the files. The software pickles the data to storage space while being transferred and then unpickles them back to allow loading again.

```
import joblib
joblib.dump(ML_pipeline, 'rf_classifier.pkl')
# To load: RFmodel = joblib.load('rf_classifier.pkl')
```

Exercice 6: Data Preprocessing

In our dataset there aren't any missing values, outliers, attributes that provide no useful information for the task. So, we could conclude that our data set is quite clean. But if we look to the output variable, the quality values represent a score between 0 and 10. Human wine preferences scores varied from 3 to 8, so it's straightforward to categorize answers into 'bad' or 'good' quality of wines. By visualizing the graph of the number of values for each category, we can see clearly that there are many bad answers than good ones. When machine learning algorithms operate digital values, we have to assign discrete values 0 or 1 corresponding to categories. We need to segment and sort data values into bins. To do this, we use The ***pandas.cu***t function useful for going from a continuous variable to a categorical variable:

```
from sklearn import preprocessing

group_names = ['bad', 'good']
catego = pd.cut(y, bins = 2, labels = group_names)
catego
label_quality = preprocessing.LabelEncoder()
# Bad becomes 0 and good becomes 1
wine['quality'] = label_quality.fit_transform(catego)
print(wine['quality'].value_counts())
print(wine['quality'].head(20))
y.head(20)
```

1. Apply the last pipeline on the transformed data and compare both results.
2. Define a new pipeline without PCA and apply it to the last data with a binary output target variable. Compare again the obtained results