PROJECT
PYTHON FOR DATA ANALYSIS

Project description

The objective is to place you in the context of a data analysis and data science project involving supervised classification. The goal is to have you practice the main steps of this type of project using real-world data.

You will work on a dataset available on ENT, and a file describing the different variables is also provided. For didactic purposes, we have slightly modified this dataset, but most of the values remain original.

It is important to note that the work you will accomplish during this project is not specifically tied to the selected dataset. Most of the steps in this work are systematically found in any data classification project. Therefore, it is recommended to thoroughly understand the purpose of each step and how to achieve it. This will certainly be useful for other projects of a similar nature.

Work to be done

The work to be carried out is divided into several steps that are characteristic of this type of project.

There are generally steps dedicated to understanding the data, preparing the data, training, and comparing different models, and then saving the best model obtained to include it in the production system.

The steps are described in the following sections. It is recommended to follow the indicated order. For each step, explanations and function usage guidance from various libraries are provided. Your task is to understand the explanations, explore the documentation of the mentioned functions, and write the code to achieve the objective of the step. At each step, make sure to verify that your code produces the expected result. To write this code, it may sometimes be necessary to rely on the code written in previous steps. It is up to you to assess which code should be reused.

At the end of the project, you will need to provide your commented code along with a clear and concise report. In particular, you should focus on analyzing the results obtained and answering the various questions posed at each step. This work must be done in pairs.

Data Import

The dataset is available on the ENT as a CSV (Comma Separated Values) file. Using the Pandas library, you need to import the file to enable data manipulation.

Make sure the import is correct by using the functionalities provided for the DataFrame object, including displaying information about the variables and their types, as well as showing the first few rows.

Data review

After importing the data, it is important to understand its structure. The better you understand the data, the more effectively you will be able to transform it to obtain accurate models.

Key aspects to know about your data generally include:

- The size of the dataset
- The type of data (numerical: int, float, or qualitative/categorical: object)
- The quality of the data (are there missing or outlier values?)
- The distribution of the data
- ...

Using the methods of the DataFrame class, examine the data and note any relevant information. In particular, it is important to identify:

- Qualitative data
- Missing data (represented by the symbol 'NA': Not Available)
- Outliers, which will all need to be preprocessed before classification (preprocessing to be carried out in Data Preparation Section).

To detect missing values, you can use the isna() method of the DataFrame class.

One way to examine the data is to create **histogram** and **boxplot** visualizations for each numerical variable.

As a reminder:

- A **histogram** shows the number of observations that fall into each range of given values.
- A **boxplot** provides a summary of the data distribution, highlighting the median, quartiles, and potential outliers.

Study the `hist()` and `boxplot()` methods of the `DataFrame` class and use them to generate histograms and boxplots for the numerical variables.

What observations can you make from these diagrams?

You may be able to identify **outliers**, which are values that fall outside the usual range for a given variable.

Data Preparation

To ensure the proper functioning of classification algorithms, it is necessary to have high-quality numerical data, meaning data that does not contain missing values or outliers.

Missing Data

While analyzing the data, you may have noticed whether a variable contains missing values ('NA').

Using the `isna()` method of the DataFrame class combined with the `sum()` function, you can determine the number of missing values for each variable.

To address the issue of missing values, several approaches are possible:

- Remove the corresponding samples
- Delete the variable entirely
- Assign a value that aligns with the variable's distribution (mean, median, most probable value, etc.)

In general, if the number of missing values is very high (more than one-third of the data), the second solution (deleting the variable) is preferred. The `drop()` method of the DataFrame class allows you to easily remove one or more columns.

During this step, you should also eliminate any columns that do not seem useful for solving the classification problem.

If deletion is not an option, the last solution (imputation) is recommended:

- For numerical variables, compute the mean or median of the variable and replace missing values with the calculated result.
- For categorical variables (often text-based), replace missing values with the most frequent category.

In both cases, you can use the `fillna()` method of the DataFrame class to perform the replacement.

Outliers

When visualizing the histograms, you should have observed if a variable takes on outlier values. It is necessary to replace these outliers with appropriate values.

Transformation of categorical variables into numerical variables

Machine learning algorithms only handle numerical values. Therefore, it is necessary to transform categorical variables into numerical variables. In the case of boolean variables, the replacement can be done directly using the `replace()` method. In other cases, it is easier to use the `LabelEncoder` class. Study this class and modify the relevant columns of the DataFrame object.

Data normalization

One last point regarding data preparation is the rescaling of variables. Generally, machine learning algorithms perform worse when the input numerical variables have very different scales. Therefore, it is useful to bring all variables to the same scale. To perform this operation, Scikit-Learn offers several methods, but we suggest using the `StandardScaler` class. Study this class and use it to normalize all the input data (which should first be transformed into a Numpy array).

After performing all these transformations, it is useful to re-examine all the variables that will be used for classification.

Search for correlations

To go further in the analysis of the data, it is important to focus on the relationships that exist between the variables. To do this, you can calculate the correlation coefficient between each pair of numerical variables using the `corr()` method of the DataFrame class. For readability purposes, since there are many variables, it is preferable to retrieve and display the correlation of a particular variable with each of the other variables.

Recall the meaning of a correlation coefficient. Which input variables are most correlated with each other? Which ones are most correlated with the output variable representing the class? In your opinion, which input variables are the most relevant for classification?

Another way to check correlations between variables is to use the `scatter_matrix()` function from Pandas. This function pairs the numerical variables two by two and displays the corresponding scatter plots. Again, since there are many variables, it is better to focus on a subset of variables, particularly those you have identified as the most promising for classification. After selecting the variables to visualize, use the `scatter_matrix()` function.

Comment on the resulting visualization.

Extraction of training and test datasets

To train a model and evaluate it, we need a training set and a test set. Creating these two sets involves randomly selecting elements from the dataset and placing them in two separate sets.

You can easily obtain these sets using Pandas functionality with a random number generator. Scikit-Learn provides ready-to-use functions. The simplest function is `train_test_split()`. There are several ways to configure this function, which takes Numpy arrays as input (for both input and output data).

Look at its documentation and then use this function to obtain the training set and the test set in the form of Numpy arrays.

What proportion of the initial dataset constitutes the training set? The test set?

Training a model

We are now ready to use the training set to train a model for binary classification. There are many algorithms for this problem, including the Perceptron algorithm that we studied in class. For this project, we propose using another algorithm based on logistic regression (also called logit).

In the Scikit-Learn library, the `LogisticRegression()` class implements this algorithm. Train a logistic regression model on the training set using this class.

It is important to understand how the algorithm works. For this, you can refer to documents available on the web. The idea is to take the principle of linear regression and apply it to a classification problem. The logistic regression algorithm is based on the maximum likelihood criterion. In the binary case, it assumes that the output

y can take two values: 0 and 1.

To verify your understanding of the algorithm, answer the following questions:

- Hypothesis: What assumption is made about the logarithm of the likelihood ratio (called the logit function): $log\frac{p(y=1|x)}{p(y=0|x)}$?
- Minimization of the cost function: What technique can the algorithm use to minimize the cost function?
- Learning: What parameters are calculated during the learning phase of the algorithm?

Model evaluation

After the training phase, the trained model can be used to predict the classes on the test set.

Make predictions on the input data of the test set.

We now aim to evaluate the model's performance by comparing the classes predicted by the model with the actual classes provided in the test set. Initially, you can simply write a loop that displays the predicted class and the actual class for each sample.

Scikit-Learn offers several metrics to obtain a quantitative evaluation of the model: `accuracy_score()`, `confusion_matrix()`, `precision_score()`, `recall_score()`, and `f1_score()`.

Review the documentation of these functions, then apply them to your test set. Recall the meaning of these different metrics and analyze the results obtained.

Improvement of the evaluation

One way to improve the evaluation of the model is to use cross-validation. It involves randomly splitting the training set into several distinct subsets, then training and evaluating the model in successive passes. In each pass, one block is reserved for evaluation, and the remaining blocks are used for training.

Scikit-Learn provides tools for applying cross-validation. The `KFold` class allows you to perform K-fold cross-validation, and the `cross_val_score()` function provides the results of the cross-validation.

After reviewing the documentation for these two tools, use them to construct a cross-validation on the dataset. The `cross_val_score()` function returns an object where the results are stored. Analyze these results and compare them with the results obtained previously without cross-validation.

Comparison with other algorithms

In general, it is not immediately clear which classification algorithm will give the best results with your dataset. It is often necessary to try several algorithms, multiple configurations of hyperparameters, and compare the accuracy of the corresponding models.

The goal of this section is to compare 4 classifiers, including 3 from the Scikit-Learn library:

- Logistic regression, previously studied
- Perceptron, studied during the course
- K-nearest neighbors (KNN)

and one classifier chosen by yourself.

Classification using K-nearest neighbors involves classifying a data point into the class of its K nearest neighbors in the feature space identified through learning. In Scikit-Learn, this algorithm is implemented by the `KNeighborsClassifier` class.

A great advantage of the Scikit-Learn library is that it provides a consistent interface for applying various learning algorithms and calculating the accuracy of the resulting models (using methods such as `fit`(), `predict()`, `cross_val_score()`...). However, each algorithm has its own set of specific hyperparameters that must be specified when constructing the classifier object. It is recommended to experiment with different configurations for these hyperparameters.

Using these interfaces, we ask you to construct a piece of code that jointly evaluates three classification algorithms on the dataset following these 3 steps:

- Training the model on the training set
- Evaluating the model using cross-validation
- Displaying the score

Saving the trained model

Once you have found a good model for your problem, you need to think about deploying it in an application. This involves saving the trained model and ensuring it can be loaded to make predictions with new data in the production system.

A model created with the Scikit-Learn library can be saved using the object serialization library called Pickle. This library provides the `dump()` and `load()` functions to respectively save and load the model. Review the documentation of these functions to understand their parameters and apply them.