HOUSE PRICES KAGGLE COMPETITION WITH SCIKIT-LEARN AND HYPER-PARAMETER TUNING

(3.5 points)

1. INTRODUCTION

In this assignment we will start practicing several machine learning basic concepts: model training, model evaluation, and hyper-parameter tuning, with the scikit-learn Python library.

The goal is to predict House Prices based on attributes such as:

- MSSubClass: The building class
- MSZoning: The general zoning classification
- LotFrontage: Linear feet of street connected to property
- LotArea: Lot size in square feet
- Street: Type of road access
- Etc.

The problem is fully described in the Kaggle competition web page: https://www.kaggle.com/c/house-prices-advanced-regression-techniques

2. DATA

Although the datasets are provided in the competition web page, you should use the file I provide in Aula Global, named "kaggleCompetition.csv". I have already carried out some preprocessing (such as missing value imputation and transforming categorical attributes into integers). In order to read it into Python, you have to use the following code:

```
import pandas
data = pandas.read_csv("kaggleCompetition.csv")
data = data.values
```

X = data[0:1460,:-1]
y = data[0:1460,-1]
X_comp = data[1460:,:-1]
y_comp = data[1460:,-1]

Once this code has been executed, (X,y) will contain the available data that you can use for model training, hyper-parameter tuning, and model evaluation. In order to participate in the competition, you will have to apply your best model to X_comp and send the predictions to the competition. y_comp contains no useful data, because the predictions for X_comp are not known. You will use X_comp just at the end of this assignment.

Given that we are going to use KNN in this assignment, it is important for input attributes to be normalized or standarized. Please, read 5.3.1 here (https://scikit-learn.org/stable/modules/preprocessing.html) in order to standarize both X and X_comp. Basically:

from sklearn import preprocessing

scaler = preprocessing.StandardScaler().fit(X)

X = scaler.transform(X)

X_comp = scaler.transform(X_comp)

3. WHAT TO DO

Summary: here, I'll give an overview of what is to be done:

- **X**, y is our available data for model training and model evaluation. X_comp is for making predictions, once our final model has been trained.
- The performance metric is RMSE.

- First of all, ¼ of (**X**,y) will be split into ($\mathbf{X_{train}}$, y_{train}) (3/4) and ($\mathbf{X_{test}}$, y_{test}) (1/4). ($\mathbf{X_{test}}$, y_{test}) will be held out and **used only for model evaluation (outer) at the end of the assignment**.
- Then, we will do the Combined Algorithm Selection and Hyper-parameter optimization **by hand**. Basically, KNN and Regression trees, with both default hyper-parameters and hyper-parameter optimization, will be evaluated with 2-fold crossvalidation (**inner**).
- Then, the best method will be selected according to their inner evaluations, and:
    - Used for training the final model (with the complete (**X**,y)) and obtaining predictions that will be sent to the competition (on $\mathbf{X_{comp}}$).
    - Used for estimating future performance (that is, to estimate what score we will get at the competition).
- Please, read the complete assignment before starting (you will need to measure times, read extra documentation from Optuna, etc.).
- Results should be reproducible

Now, the detailed steps to be done:

3.0) Read and understand the scikit tutorials I have left for you in Aula Global under **BASIC SCIKIT-LEARN USAGE** (Intro to decision trees and scikit-learn, Evaluating decision trees with Holdout and crossvalidation, and Hyper-parameter tuning for decision trees).

3.1) (0.2 points) Evaluate a DecisionTreeRegressor and a KNeighborsRegressor with **default** hyper-parameters.

3.2) (0.2 points) Now, use **random-search** for tuning the following hyper-parameters of DecisionTreeRegressor:

- *min_samples_split:* this is tipycally an integer that may range from 0 to the maximum number of instances. However, scikit-learn allows to use a proportion that is a real number and ranges from 0.0 to 1.0. Use this hyper-parameter in the second way (as a real-valued fraction).
- *criterion* (mse or friedman_mse).
- *max_depth:* integer
- Check the following web-page for info on the hyper-parameters: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html

3.3) (0.2 points) Use **random-search** for tuning the following KNN hyper-parameters:

- *n_neigbors*
- *weights*: (this is the type of distance: euclidean or inverse euclidean)
- *p*: 1 or 2 (this is the exponent of the distance)
- Check the following web-page for info on the hyper-parameters: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html

3.4) (0.5 points) Use now **scikit-optimize** (skopt) under the same conditions of 3.2-3.3. Use the same number of iterations or budget, so that you can draw some conclusions about what hyper-parameter tuning method obtain better results or is faster (or both).

3.5) (0.9 points) Use now optuna under the same conditions of 3.3-3.4. Try to use the same computing power (iterations or

budget). Please, read also **3.9)** in the assignment, in case you want to develop this point further.

3.7) (0.25 points) Determine now what is your best method according to their **inner** evaluation. You can also draw some conclusions: has hyper-parameter tuning being able to improve KNN and/or Decision Trees? What HPO method is best? What HPO method is faster?

3.8) (0.25 points) Now, make an estimation of the performance you will get at the competition. This involves evaluating the selected model on ($\mathbf{X_{test}}$, y_{test})) partition, that was excised at the beginning just for this purpose.

3.8) (0.4 points) Then train the final model using the selected method on the whole available data (**X**,y) and use it to make predictions on $\boldsymbol{X_{comp}}$. Register to the competition, send your predictions, and obtain a **screenshot** of the place (rank) you got at the competition. **IMPORTANT!!!** In the dataset you will find in Aula Global, the response variable *y* is the log of the *Sale Price*. (*y=log(SalePrice)*), because the competition required models that predicted well *log(SalePrice).* However, the competition requires that you send actual prices (not logarithms). Therefore, once you have computed *y_comp = your_best_model(**X**$_{comp}$)*, you will have to compute *SalePrice = exp(y)*. It is SalePrice predictions that you have to send to the competion.

3.9) (0.6 points) These points are reserved for using optuna beyond what I have explained, and it should be an original contribution from the student. For instance, using Optuna plot utilities to visualize the search and draw conclusions. Or exploring the hyper-parameter space in more complex ways. Etc.