Comparision of Sci-kit-learn and ML.NET regression

Mateusz Kryszczak
Faculty of Electronics Engineering
Wroclaw University of Science and Technology
Wroclaw, Poland
219664@student.pwr.edu.pl

I. Introduction

With the fast grown of the data, including the internet of things data, machine learning methods are getting more and more popular. One of the machine learning tasks is the supervised learning. It consists of classification, regression and clustering. The regression is one of the most commonly used for data analysis – from mathematical point of view, it include set of statistical processes for estimating relationship among observations and therefore might be used for making predictions.

II. Scope of work

The main objective is to find at least one regression algorithm with is present in both frameworks and compare it.

The scope of work include:

1. research about considered problem,
2. choosing appropriate dataset,
3. dataset analysis,
4. find optimal parameters for considered regression method, for example using Grid Search algorithm,
5. dataset preparation: dataset splitting, saving into separate files (train and test parts) for further usage in both implementations,
6. implementing the testing program using Python 3, Sci-kit-learn and other necessary dependencies,
7. implementing the same program using .NET Framework and ML.NET
8. conclusions and analysis of research results,
9. preparing the final documentation.

Both regression implementations (in Python and .NET) do the following:

1. Load given dataset to memory (train and test splits)
2. Train regressor,
3. Perform K-fold cross validation with train dataset,
4. Evaluate results with test dataset,
5. Export output data to csv file for further analysis.

Separate Python program was created to prepare the dataset (split it into train and test subsets, encode labels into binary features and save it into two csv files).

The comparing parameters will be the R-squared and RMS (root mean square) values.

III. Framework comparision

Today's market offer plenty of machine-learning dedicated libraries, some of them include sci-kit-learn, Keras, Theano or Google's TensorFlow. These libraries or frameworks offers application programming interfaces in wide variety of languages, but definitely Python is the most common one. Other are C++, R, F#, C#, JavaScript, Java [2].

One of the most popular libraries widely used by mathematicians, researches, students and even hobbyist – due it's simple application programing interface is sci-kit learn. It was first released in 2007, it is implemented in Python programming languages, and has dependencies in other well-known libraries for statistical data processing like NumPy and Pandas. It also offer fast and reliable way for plotting the data – the Matplotlib [3].

From the other hand, Microsoft released it's own package dedicated for machine learning – ML.NET. The library itself its fairly new – it was released in may 2018, therefore it has only few algorithms implemented, but still it allows user to perform fully customizable experiments. Because C# is strongly typed language (Python is not) it might require slightly more lines of code to achieve the same results, but once it's done, the model usage is more user-friendly and more prone to mistakes. The library is dedicated for business intelligence processes, to be used with another Microsoft web frameworks, like ASP.NET i.e. for e-commerce sales prediction within some period of time basing on data from previous months or years [4].

Table 1 Available regression models

Sci-kit-learn	ML.NET
• Decision Tree Regressor • Ordinary Least Squares • Ridge Regression • Lasso • Logistic Regression • Stochastic Gradient Descent • … and more	• Fast Tree and Fast Tree Tweedie regressors • Online Gradient Descend • Poisson Regression • Stochastic dual coordinate ascent

In the table 1 available regression models for both frameworks was shown. First observation is such, that SK-learn offers a lot more models, but also they are generally simpler. Most of them are a basic, well known models. The ML.NET offers more advanced algorithms, but in smaller amount.

IV. Dataset Analysis

Chosen dataset is housing. This dataset is a modified version of the California Housing dataset available from Luís Torgo's page (University of Porto). Luís Torgo obtained it from the StatLib repository [5].

The dataset has he following features:

1. *longitude*: A measure of how far west a house is; a higher value is farther west (float)
2. *latitude*: A measure of how far north a house is; a higher value is farther north (float)
3. *housingMedianAge*: Median age of a house within a block; a lower number is a newer building (float)
4. *totalRooms*: Total number of rooms within a block (float)
5. *totalBedrooms*: Total number of bedrooms within a block (float)
6. *population*: Total number of people residing within a block (float)
7. *households*: Total number of households, a group of people residing within a home unit, for a block (float)
8. *medianIncome*: Median income for households within a block of houses (measured in tens of thousands of US Dollars) (float)
9. *medianHouseValue*: Median house value for households within a block (measured in US Dollars) (float)
10. *oceanProximity*: Location of the house w.r.t ocean/sea (text)

Dataset has 20641 observations.

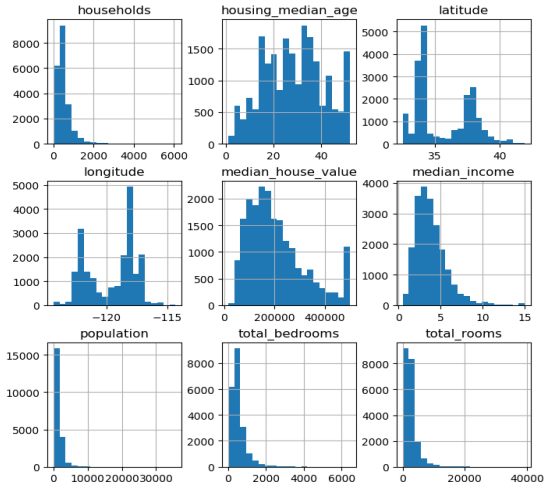

Figure 1 Dataset histograms

On the fig 1. Histograms of all the features was shown. It is easy to notice, that most of the features has Gaussian distribution, but they are right-skewed. Exceptions are latitude and longitude – because these values are not statistical, they depends on geographical location.

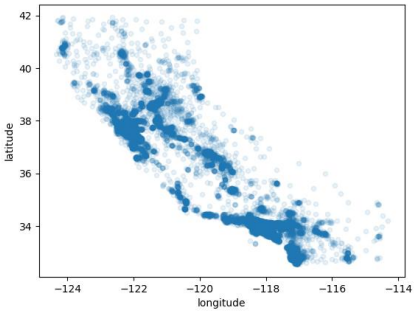

Figure 2 Population density in the function of geographical location

The figure 2 present the geographical location of considered households. The shape of the points is actually California's borders and the main clusters of points are San Francisco and Los Angeles.

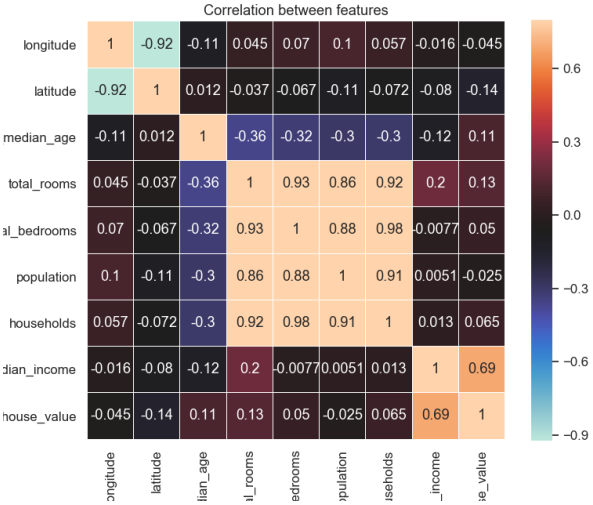

Figure 3 Feature correlation heatmap

Heatmap presented on figure 3 shows features correlation.

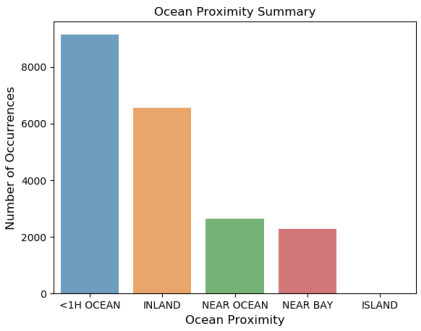

Figure 4 Ocean proximity feature

The ocean proximity feature (fig. 4) is the only text feature, thus it needs to be binarized by splitting it into 5 sub-features (one-hot-encoding).

V. Dataset preparation

The dataset will be consumed by python and .NET implementation, thus a proper preparation is mandatory.

First, the *ocean proximity* feature text feature was binarized using one-hot-encoding. For each of it's possible text value, an new dummy, binary column was added. This operation will ensure equal weight for each value. The original *ocean proximity* column was dropped [6].

Then we are dropping all the observation containing not-a-number values. Finally we are splitting the whole dataset into training (70%) and testing (30%) subsets with random seed. Both subsets are saved into separated .csv files, so it is possible to use it multiple tames with same randomness seed in both implementations.

VI. Comparision

Decision tree regression model will be compared, because it is the only one which is present in both libraries.

For hyperparameter tuning an grid-search algorithm was utilized. Because only Sci-kit-learn offers grid search (Microsoft offers it for now only as azure service together with more ML models) it was tuned only once on Python application. Then the parameters were used in both implementations.

Train set was divided into 10 folds. Program flow is as follows:

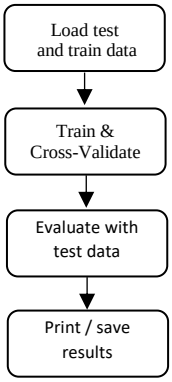

Table 2 Default decision tree regressor parameters

Parameter	SK-Learn	ML.NET
max_depth	None	100
min_samples_leaf	1	10
max_leaf_nodes	None	20

Table 2 presents default parameters for both libraries (only the parameters which are common for both libraries were chosen).

At first both implementations were run using its default hyperparameters.

Table 3 Metrics comparision for default model parameters

Metric	SK-learn	ML.NET
Cross-val. R^2	0.631	0.827
Cross-val. std. deviation	0.03	0.06
Evaluation R^2	0.66	0.83
Evaluation MSE	45.54 E8	22.99 E8
Train time *[ms]*	1902	5359

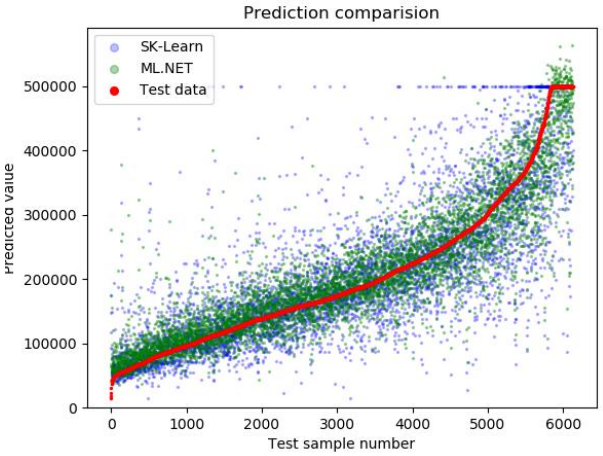

Figure 5 Prediction results for default parameters

Table 3 and fig. 5 presents metrics comparision and prediction results for both implementation. The prediction was done using the test observations (not used in learning). In this case, the ML.NET regressor outperformed the Sci-kit-Learn.

The second benchmark was utilized using grid-search tuned hyperparameters. It was done with assumption, that ML.NET and Sci-kit-learn are using same scaling on hyperparameters.

Parameter	Value
max_depth	10
min_samples_leaf	1
max_leaf_nodes	19

Table 4 Metrics comparision for optimized hyperparameters

Metric	SK-learn	ML.NET
Cross-val. R^2	0.633	0.677
Cross-val. std. deviation	0.03	0.02
Evaluation R^2	0.64	0.68
Evaluation MSE	47.95 E8	43.01 E8
Train time *[ms]*	670	3617

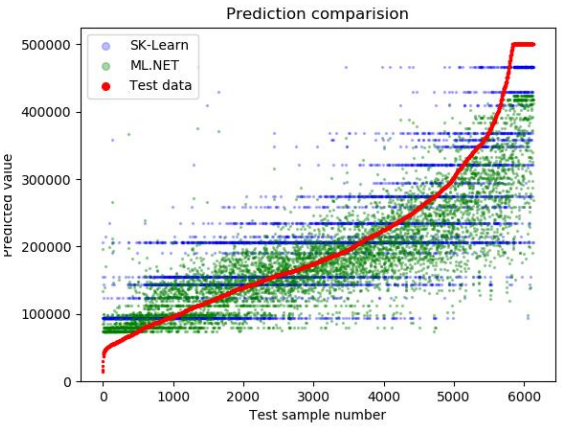

Figure 6 Prediction results for tuned hyperparameters

Figure 6 presents results of test using tuned hyperparameters. The prediction seems to be underfitted, that might be because grid search algorithm found local minimum. Same for the ML.NET – here, the R^2 score is much worse than before – probably is it due to different hyperparameters scaling in library implementation. Also by looking at the Sci-kit-Learn predictions, its easy to spot decision borders.

VII. Statistic test

After obtaining prediction values from both regression implementations, it is wise to perform statistical test – in this case it is independent two-sample Student's t-test.

H_0 hypothesis might be stated as follows: "*there is dependency between Sci-kit-learn and ML.NET predictions"*.

	p-value (probability)
Default regression model hyperparameters	72%
Tuned regression model hyperparameters	<0.01%

For the default model parameters H_0 hypothesis cannot be rejected (test was performed with confidence level of 95%) – there is 72% of probability that hypothesis is correct.

For tuned parameters it is possible to reject the hypothesis with confidence level of 95%). This indicates that there is no dependency between two regression implementation in that case.

This test was performed with assumption, that both prediction values has Gaussian distribution.

VIII. Summary

The regression and prediction were successfully implemented using ML.NET Sci-kit-Learn libraries. The benchmark test covered decision tree regression model. Using the default parameters provided by implementations, the ML.NET outperformed Sci-kit-Learn significantly. Then, the grid search algorithm was utilized to tune algorithm hyperparameters. The second test was done using tuned hyperparameters. The difference between R^2 parameter for default and tuned parameter (using Python SK Learn) is nearly the same (0.633 vs 0.631), but the evaluation score is slightly worse, because of underfitting. The ML.NET also performed worse in this case – this is probably due to different hyperparameters scaling in its implementation. For now, ML.NET does not offer of the shelf parameter offline tuning algorithm (like grid-search).

The Sci-kit learn library is dedicated for broader range of people, including mathematicians, researchers and hobbyist. It offers a lot of fundamentals statistical models, unlike the ML.NET library, which is dedicated for business purposes. It has only few, rightly selected regression models.

IX. References

[1] W. Richert, L. Coelho, Building Machine Learning Systems with Python, Packt Publishing, 2013

[2] https://en.wikipedia.org/wiki/Comparison_of_deep _learning_software

[3] https://scikit-learn.org

[4] https://docs.microsoft.com/en-us/dotnet/machine-learning/

[5] https://www.kaggle.com/harrywang/housing/home

[6] S. Raschka, Python Machine Learning, Packt Publishing, 2016