Imperial College London

Introduction to ML

Artificial Neural Networks

Coursework 2

07/11/2022 - 25/11/2022

MSc Computing in Artificial Intelligence
MSc Computing in Artificial Intelligence & Machine Learning

Authors :

Pierre-Antoine ARSAGUET
Email: pea22@ic.ac.uk
CID: 02288078

Louis BERTHIER
Email: ldb22@ic.ac.uk
CID: 02285087

Nikita DMITRIEFF
Email: nnd19@ic.ac.uk
CID: 01776095

Ryan EL KHOURY
Email: re122@ic.ac.uk
CID: 01765361

Contents

List of figures

Chapter A

Presentation of the coursework

A.1 Introduction to the dataset

For this coursework, the **California House Prices Dataset** is analyzed.
It is made up of 10 variables and more precisely of **9 features and 1 output**.
This is a linear regression problem where from the observations of the 9 features, a model must be trained and able to predict a value associated with the output variable. The variables are as follows:

- **Longitude:** longitude of the block group
- **Latitude:** latitude of the block group
- **Housing median age:** median age of the individuals living in the block group
- **Total rooms:** total number of rooms in the block group
- **Total bedrooms:** total number of bedrooms in the block group
- **Population:** total population of the block group
- **Households:** number of households in the block group
- **Median income:** median income of the households comprise in the block group
- **Ocean proximity:** proximity to the ocean of the block group
- **The output variable to be predicted:** Median house value (of the houses of the block group)

This dataset contains exactly **16512 observations**. As far as the number of features is concerned, this dataset **does not have the curse of dimensionality**. All features are represented by **numerical data in float form** with the exception of the last one, *'Ocean proximity'*. This is a problem that needs to be dealt with in order to take this feature into account when developing the model.
Another problem is the **absence of data for some of the observations** in the *'Total bedrooms'* feature. More precisely, **168 data are missing** which represents only **1%** of the whole dataset.

Firstly, the pre-processing and the model used will be introduced and justified in relation to the parameters used. Then this model's training and evaluation methods will be presented before discussing the search for the best hyperparameters. Finally, once the search for the best hyperparameters has been carried out, the best model obtained will be introduced with the associated results.

A.2 Description of the pre-processing

To implement machine learning, the most important step is the pre-processing of data before any injection into a neural network following the **garbage in - garbage out principle**.
The pre-treatment process is as follows :

1. Firstly, the dataset must be split into 3 datasets, namely **train, validation, and test** (60%, 20%, 20%). The last one is kept only to apply the best model obtained after its evaluation and the search for hyperparameters. It is important to note that the data pre-processing operations must be **applied to each dataset but in a local and not global way** in order not to bias them.

2. As stated above, some data are represented by strings and need to be converted into numerical data in order to integrate them into the neural network. To do this, they are encoded using a **one-hot encoder** (LabelBinarizer() from sklearn, keeping in memory how the data are fitted) which will add as many columns as there are different values for the feature (here 5). The columns created will thus be filled with 0 and 1 depending on the observations.

3. Also, to fill in the missing values, these have been **replaced by the median value**. Indeed, the feature *'Total bedrooms'* has only 1% of missing data, so it is unthinkable to drop this feature when **99% of the available data is valid and usable**. Also, the median was chosen over the mean for consistency with the data. As shown in appendix .1, the distribution has **many outliers** (precisely 1016) and the mean is therefore no longer as relevant.

4. It is remarkable that the different features have completely different scales with large amplitudes ranging from a few hundred to tens of thousands. To overcome this problem, it is important to **normalize the data** in order to put the data on an **equal footing** by bringing them to the same scale (or to scales of similar orders of magnitude) and to **speed up the processing within the neural network**. To do this, the **max-min normalization** defined by equation A.2.1 is applied to the data.

$$x_{\text{normalized observation}} = \frac{x_{observation} - x_{min}}{x_{max} - x_{min}} \tag{A.2.1}$$

   Normalization by Z-score was not appropriate since the features have extreme outliers among the observations. Indeed, graphs similar to the one in appendix .1 are obtained for the features *'total rooms'* and *'households'*. Despite the presence of anomalies, the features are globally **distributed in a uniform manner** over a fixed range, which justifies our choie. However, **normalization by clipping** could also have been interesting for features where the outliers are significant and numerous.

5. A study of the **correlation between the different features** is presented in appendix .2. The idea was to study the impact of the features on the output variable. However, as the table shows, even if some variables are much less important than others, it is **better to use the available data**. There is **no need to remove a feature** for prediction and the data is of good quality so we might as well use it to develop the neural network.

6. Rather than using the library implemented in part 1 of the coursework, the authors decided to **use the PyTorch** to implement the model. Thus there is an additional step of transforming the data frames into PyTorch's tensors.

A.3 Presentation of the model

The model implemented with PyTorch is a **multi-layer linear network**. According to the literature and the task at hand, namely linear regression, the **ReLu activation function** is used within the neurons. This is one of the most common functions for regressions and also one of the best.

The learning rate and the optimizer were set arbitrarily. Although some papers agree that SGD generalizes better, others claim that Adam performs better in terms of time and reliability of convergence. The advantage of **selecting Adam** is also that it has an **adaptive learning rate**, i.e. it computes individual learning rates for different parameters.
The chosen learning rate is 10^{-4} and is derived from various personal projects as well as from the state of the art which **ensures a good learning for the model in a rather small number of epochs**.

The method of evaluation of the model is introduced in section B.1 Other parameters called hyperparameters are variable in order to seek to optimize them. They are introduced in section B.2.

Chapter B

Model Tuning

B.1 Description of the evaluation

The model is evaluated using the **RMSE**. The latter allows errors to be **heavily penalized** (unlike the MAE) and an error to be expressed in the **same unit as the predictive value** (unlike the MSE). Alongside this metric, the **coefficient of determination** (R^2) is also calculated to judge the quality of the regression obtained with the model by measuring **how well the model fits the data** used for prediction.

B.2 Hyperparameter search

A model can be improved by looking for optimal values of hyperparameters. The hyperparameters considered are the following:

- **The batch size** represents the number of training examples utilized in one iteration. It takes 3 values: **[128, 256, 512]**. A batch is defined in equation B.2.1.

$$\text{batch} = \frac{\text{Number of observations}}{\text{Batch size}} \tag{B.2.1}$$

- **The number of epochs** which defines the number of times the model is trained on the training set. It takes 3 values too: **[50, 100, 150]**.
- **The number of hidden layers and the number of neurons** that stands for the depth and complexity of the model. 4 architectures are proposed, namely:
  - **3 hidden layers** with the following number of neurons per layer: **[256, 512, 128]**
  - **4 hidden layers** with the following number of neurons per layer: **[256, 512, 256, 128]**
  - **5 hidden layers** with the following number of neurons per layer: **[256, 512, 512, 256, 128]**
  - **6 hidden layers** with the following number of neurons per layer: **[256, 512, 1024, 512, 256, 128]**

To ensure the stability of the different model combinations, **cross-validation** is performed with **5 replications and data shuffling** with training on the training set and performance measurement on the validation set. The best model is therefore the one that obtains the **best average RMSE among the 36 possible model combinations** in this hyperparameter search.

Chapter C

Final evaluation of the best model

C.1 Identification of the best parameters

The results of the cross-validation and hyperparameter search are shown in figure 1.

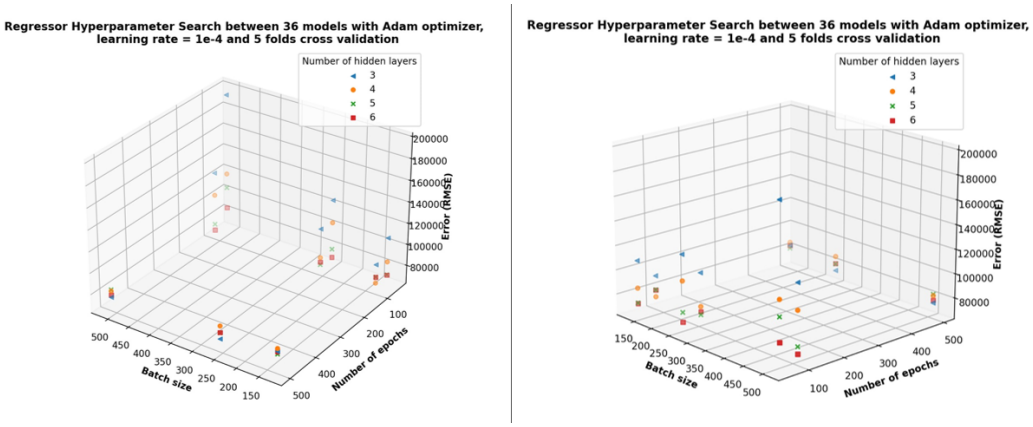


Figure 1: This graph represents the evaluation of the 36 models through the search for hyperparameters. These results are obtained with the Adam optimizer, a learning rate of 10^{-4} and the RMSE is an average obtained after a 5 folds cross-validation. **Left:** View 1 **Right:** View 2

With this search for parameters several points should be noted. First of all, this search for parameters is **simplistic** since the hyperparameters only take 3 to 4 different values and only 3 hyperparameters are considered. For example, it would have been interesting to also **vary the learning rate**, to **extend the possibilities of the hyperparameter values** to 10 to cover a wider field of action or to **increase the number of repetitions of the cross validation** to ensure the stability of the different models.
However, the aim of this coursework is not to fundamentally find the best combination but to **understand how the parameters interact and to have a critical opinion on the results obtained**. Especially as the **simulations were demanding in terms of physical resources**, so increasing the computational load was impossible.

It can be observed that for few epochs and a low number of layers, the model has difficulty in capturing the complexity of the data and therefore the error is huge. These are **cases of underfitting**. The same results are more or less obtained for 100 epochs except that the models with 4 layers have this time a much lower error. It testifies to the learning of the **models that are no longer underfitting**. The case with 500 epochs shows the **problem of overfitting** of almost all the models since the error does not decrease compared to 100 epochs. On the contrary, it stagnates or even worse, the error increases.

If the batch size becomes small then the batch increases and therefore the model has **more data to train on** but here the variation of the batch size does not seem to have a big effect or at least for the 3 possibilities that were indicated during the hyperparameter search. In the same way, having **more epochs allows the algorithm to deepen its training** and therefore to develop better until overfitting. Similarly, the deeper a network is (and therefore has more neurons), **the better it will be at capturing the relationships in the data** that are fed to it, if there are enough data. Furthermore, the literature agrees that it is **more interesting to have a deep model with fewer neurons per layer than a shallow network with many neurons**.

However, the task is relatively simple (linear regression) and the number of data and their complexity are not very high (about ten thousand observations with only 9 features) compared to tasks where it deals with millions of observations and hundreds of features. It is therefore **not necessary to have a very deep network** (with dozens of layers) or **to have a very high number of neurons** (with thousands of neurons per layer). Also, it is important to be careful about generalisation. A model that is too complex will **memorise and not generalise, which results in overfitting** (bias-variance tradeoff). Another important point is the resource and computational time, it is often more interesting to **limit oneself to a simpler model even if it means losing a little in performance to gain a lot in computational time** (time-performance tradeoff).
Finally the **model with the best performance** is obtained with the following hyperparameters:

- **Batch size** = 256
- **Number of epochs** = 50
- **Number of hidden layers** = 6 so $[256, 512, 1024, 512, 256, 128]$

C.2 Results obtained with the final model

With the best parameters indicated in section C.1, the following scores are obtained on the test set:

- **RMSE** = 68294.286
- **R²** = 0.63

With cross-validation, overfitting is limited, but it is possible to implement other methods to ensure that **overfitting is reduced as much as possible**, in particular by adding a **dropout layer at the end of the neural network** to randomly remove neurons during training. Another method could have been to add an **early-stopping parameter** that would stop the training after a certain number of epochs to be defined if the error (loss function) continued to rise or stagnate.
The loss curves of the best model are shown in figure 2.

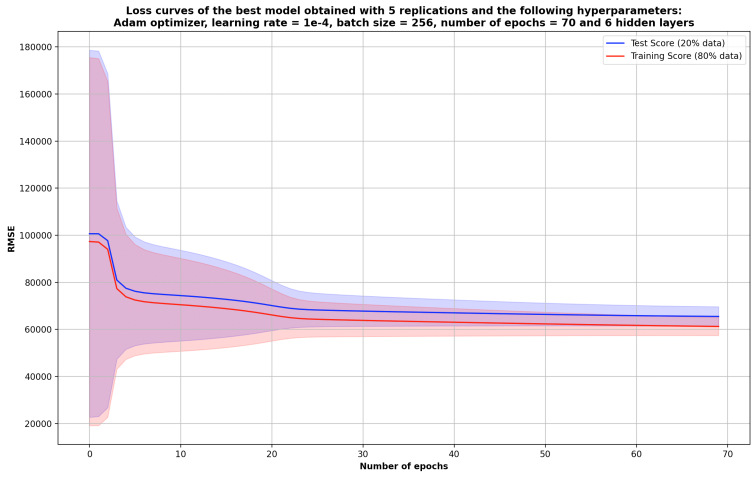
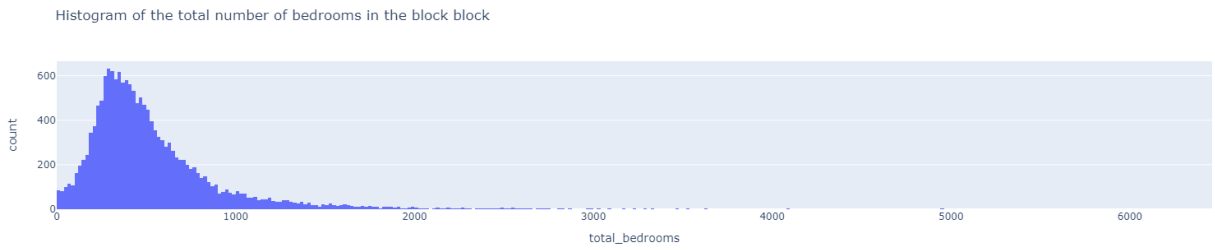


Figure 2: This plots show the evolution of the loss curves of the best model defined in section C.1 with 5 replications and 70 epochs instead of 50 to show the beginning of overfitting. The dark lines are the mean and the shaded areas around them are the variance associated with each of the means.

Unsurprisingly, there are **better results during training than during testing** (with a small gap) and a curve that **starts to stagnate around the 70 epochs**. This shows that the **model generalizes** (and does not memorize) and it justifies the impact of the cross validation which allows to select the model with 50 epochs and not 100 or 500 where there is indeed overfitting.
The RMSE and R² are high despite the search for optimal hyperparameters but **remain decent**. As previously stated, several points could have been addressed or implemented to have more possible combinations (models) or to limit the overfitting of these different combinations.

Appendices

.1 Distribution of the total number of bedrooms



.2 Correlation between the different features

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
longitude	1.000000	-0.925271	-0.112511	0.042470	0.067685	0.097077	0.052289	-0.016720	-0.049501
latitude	-0.925271	1.000000	0.015270	-0.031978	-0.063344	-0.104968	-0.066832	-0.077053	-0.139312
housing_median_age	-0.112511	0.015270	1.000000	-0.363465	-0.322462	-0.299436	-0.305403	-0.115784	0.105657
total_rooms	0.042470	-0.031978	-0.363465	1.000000	0.930434	0.855189	0.916567	0.193670	0.131024
total_bedrooms	0.067685	-0.063344	-0.322462	0.930434	1.000000	0.876023	0.978947	-0.009871	0.046617
population	0.097077	-0.104968	-0.299436	0.855189	0.876023	1.000000	0.905950	0.001234	-0.026755
households	0.052289	-0.066832	-0.305403	0.916567	0.978947	0.905950	1.000000	0.010145	0.062973
median_income	-0.016720	-0.077053	-0.115784	0.193670	-0.009871	0.001234	0.010145	1.000000	0.689618
median_house_value	-0.049501	-0.139312	0.105657	0.131024	0.046617	-0.026755	0.062973	0.689618	1.000000