Prediction Exercise

Rita Pessoa Correia

March 19, 2021

1 Programming Language and Development Environment

The programming language that I decided to use was Python, because it's one of the languages that I am more comfortable with and is also a good language for handling machine learning and deep learning algorithms.

As the development environment, I used Google Collab Notebook, in order to have the possibility to use Google's GPUs and also because of its block-based environment that allows to run and test parts of the code individually, in an intuitive and easy way, without having to install `pip` dependencies and required packages.

2 Data Pre-processing

In the first place, I created two variables (`df_test` and `df_train`) in order to store the two given datasets. Then, through the `concat()` method, I joined both files into one (`X`), so I could initially pre-processing the nominal categorical values in both files, turning them into numeric parameters. This type of values are non-measurable (unordered) data, so if both files were pre-processed separately, the same categorical values could turn into a different numeric ones in each file. Then, I allocated the last column of `X` (that contained the VTR value) to a new variable called `Y`. As I know that the last 2500 rows of both `X` and `Y` belongs to the test set, I was able to separate the correctly, later.

Then, I casted all of the categorical values of `X` to `str` type, so they can be handled properly and with no errors. Next, I had to handle `NaN` values, that in this case only existed in the "domain" row. For that purpose, I used the `SimpleImputer` method from `sklearn.impute` library, with the `most_frequent` strategy.

After all the `NaN` spaces were filled, I had to transform the textual (or categorical) predictors into numerical variables through the `LabelEncoder()` method from the `sklearn.preprocessing` Python library. Then, and because these are non-measurable values, it was necessary to create some dummy variables, so a parameter assigned with a greater number, doesn't have more weight than the ones assigned with a lower one. For this purpose, I used the `get_dummies()` function of `pandas` library, because that function can turn a categorical variable into a series of zeros and ones, which makes them a lot easier to quantify and compare.

Finally, I used the `StandardScaler()` method to standardize all the parameters by subtracting the mean and then scaling to unit variance, and split up again `X` and `Y` into train and test sets, as it was before (the first 1000 samples for training and the rest for test).

3 Deep Neural Network

To solve this regression problem, I decided to create a Deep Neural Network (DNN) with 6 hidden layers, as we can see in the figure 1.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 512)               1774592
_________________________________________________________________
dense_7 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_8 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_9 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_10 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_11 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 129
=================================================================
Total params: 2,283,905
Trainable params: 2,283,905
Non-trainable params: 0
```

Figure 1: DNN model summary.

Because it's a regression problem and the output needs to be a real value and not a range of values, I decided to use a `linear` activation function in the output layer, and `ReLU` in the other layers, in order to allow the model to learn faster and perform better.

Then, I compiled the model with the Mean Squared Error (MSE) loss function (which is the default loss function for regression problems) and `Adam` optimizer. For the training process, I defined 30 epochs with a `validation_split` of 0.3.
The figure 2 represents the graphic that compares both training loss and validation values with MSE for epoch.

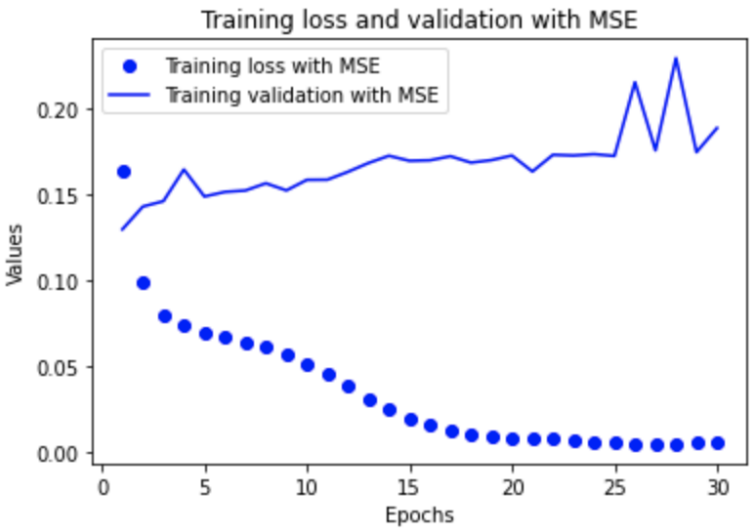

Figure 2: Training loss and validation with MSE.

Finally, I made the prediction with the `predict()` method and saved the results in a new CSV file using the `DataFrame()` method from `pandas` to firstly transform the prediction into a DataFrame object and then `to_csv` method to actually save it.