Machine Learning For a House pricing Prediction Web Application

Alex Soudant

Ynov Ingésup M1,

20 Boulevard Général de Gaulle, 44200 Nantes

Correspondence: Alex Soudant. E-mail: alex.soudant@ynov.com

report on the 1st of March 2017

Our planned objectives were to run the tensorflow algorithms for deep learning on housing adverts from Nantes area first.

A simple model to try is the linear regression model. To achieve this objective, I started by divided the dataset of images into two subsets : the training set of 457 images and a validation set of 196 images. Due to the small amount of data, no test set was created. After that, I also deleted adverts that had no price displayed, which resulted in suppressing eleven images in the training set and 6 in the validation set. i could then build the function graph and run graph sessions to test different loss calculation and optimizers. The selected loss calculation represents the absolute difference between the price estimation and the real price from the training set averaged over all 457 adverts. After trying differents optimizers, the momentum optimizer gave slightly better results (but not significantly) compared to gradient descent and adagrad. Finally, when trying to set the learning rate at best as I could, I obtain a maximum of 39.9% of accuracy in training and 36.7% in validation. These results shows that a simple linear regression model over the dataset I am using is not enough to predict with confidence housing prices just by using pictures as I first expected (see TensorFlowHousingLinearRegression.ipynb).

As suggested by Jeff Abrahamson, I then proceeded to create new images by "jiggling" the original images in a way that only a few pixels change between replicates. With this method I obtained a new training set of 1784 images and a new validation set of 760 images. However, the results are quite close to the linear regression model without this operation : 39.4% accuracy for training and 37.2% for validation (see ImageJiggle.ipynb).

I then adapted the tensorflow algorithm to predict prices with a logistic regression model. This model have the avantage to predict only categories of prices instead of a continuous pricing variable. In that purpose, I modified accordingly the prices labels to 10 categories based on quartiles calculation. By using this model, the accuracy sharply increased over the training dataset with a maximum of 86.1 % bu the validation accuracy dropped to 12.1%. Therefore this model cannot predict accurately prices when pictures outside from training are used (see TensorFlowHousingLogisticRegression.ipynb).

Another approach to increase the accuracy of the linear model is to consider adding supplementary information about the houses I have pictures from. To evaluate this potential, I created a dataset from the json files obtained when scrapping to test if the number of rooms, the surface of the house, the surface of the land and the number of pictures posted through the advert could have an impact on predicting prices. So I did

not include images at first to test the model and the results show a maximum accuracy of 45.2% on training and 13.1% on validation when including all four variables (see TensorFlowHousingImages.ipynb). Still validation fails to be satisfactory but it seems that some information can be extracted from these house features.

As this approach looked promising, another way to improve it is to add a hidden layer to the model. The idea behind this type of modelisation is to run at first the model through the training dataset and send a first set of price estimations to a second layer that will try to improve these first results. With this methodology, I could improve the price predictions accuracy to 63.8% for training and 23.9% for validation (see TensorFlowHiddenLayer.ipynb).

Now that one hidden layer has proven to be useful, what could two layers could do ? Trying two hidden layers did not improve again the results unfortunately : accuracy of 65.9% for training and 19.0% (see TensorFlow2HiddenLayers.ipynb).

Now that we know that one hidden layer on house features can be informative on prices estimations, we can merge features and images into the same analysis. To keep a good processing rate, I had to build batches of data instead of the full dataset that will be computed in a parallelized way. Running this model permitted to reach 68.3% accuracy for training. However, I could not obtain any interpretable validation accuracy as I only obtain negative values (see TensorFlowHiddenAndImages.ipynb).

I think this is already quite a lot of material to present for an oral presentation. I still need to build clean examples of the code I built. It is clear that I am limited by the size of the dataset I am using but maybe running longer computation with different parameters could achieve a better accuracy.