Machine Learning For House Pricing Prediction in Loire Atlantique Region

Alex Soudant

Ynov Ingésup M1,

20 Boulevard Général de Gaulle, 44200 Nantes

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

Abstract
I present here a project to generate predictions for housing prices in Loire Atlantique Region (France) based on pictures analysis from sale adverts. I used the openCV librairy for computer vision to prepare pictures followed by the use of Tensorflow library for deep learning analysis. The goal is to produce an easy and fast way for potential house buyers/sellers to estimate a fair price for properties before contacting real estate agents. Results show that predictions based solely on pictures are not accurate enough to predict prices of properties on sale. I tested different approaches to improve prediction accuracy that will be detailed further in this report. The best set up so far was to build a neural network with both pictures and house features (i.e. number of rooms, house surface, land surface and number of pictures) as parallel neurones on the same layer then merge the resulting tensors in an output layer. To gather pictures and features necessary to this analysis, a scrapper was built using Ember framework and ran over a house sale agency website. About 600 adverts were obtained for Nantes city which represent over 3000 pictures. Angers and Brest city were also scraped but not used in training or validation for the model.
Key words : Machine learning, Computer vision, Housing price, prediction

Introduction

Housing is an important market for business purposes. Nowadays, potential buyers/sellers for properties have to manually visit a high number of housing agents websites to be able to estimate a range of price. This is a complex and time consuming process and the information obtained may not be accurate. It is also known that the difference in prices between websites for the same property can be quite high and may cause a lack of understanding of the real market prices. Hence I propose to develop a machine learning tool to assist individuals into finding a more understandable price from real estate.

Data Obtention

To our knowledege, at the time of proposal writing, there was no public dataset available already published to fulfill such a study on the determination of housing price based on property prictures. This is due to the fact that sell prices are private in France so we cannot use recent archives of estate pricing. Therefore, I built a javscript scraper using Ember framework to help building a decent dataset to perform analysis.

Data Preparation and Cleaning

After running the scraper to gather pictures from the website into different advert folders in the hard drive, I noticed that some pictures were corrupted. The main consequence in such pictures was that the resolution was pretty poor and uneven thoughout the picture. Therefore, it can be expected that these pictures contain less information than a correctely downloaded picture which can greatly affect the analysis when training the model to predict prices. A easy way to get rid of these images is to read the size of the picture by using opencv library and deleting every picture below a threshold of 9000 octets. To produce tensors based on image information, the original pictures need to be squared and of a size that will not impair too much computer performance. Therefore, all pictures were fisrt cropped to take only a central square part for analysis and then resized to obtain a resolution of 58 pixels for width and height. As I did not want to introduce color analysis for price estimation, all pictures were also converted to a gray scale which also help reduce picture size and so training speed. A last technique used on pictures is to artificially increase their numbers by swapping some pixels between each other and creating different pictures for a computer vision point of vue. To try this technique, I created three copies from the first image of each advert. The first copy have 58 pixels swapped randomly over the whole picture, the second copy is flipped horizontally and finally the last copy have both the 58 pixels randomly shuffled and all pixels flipped over. Features of the houses on sale, extracted from the online adverts, were saved into one json file for each advert. Therefore, the first quality check was to assert if all advert folders had a json file and delete folders that did not contained one. Once, I was sure to have in each advert folder, a set of pictures and a json file, the features for every advert were regrouped into a csv file in a table format. Features and pictures were then paired into pickle files following tensorflow tutorial recommandations. After having each picture paired with features, I then checked for prices and house surface equal to zero and removed these data lines before analysis. Adverts with no price indicated seem to match properties that need to be directly negociated by contacting the real estate agency. When no house surface is indicated, a rapid check on the real estate website shows that some adverts concern land properties with no built house on it. Due to the fact that the scraper gather adverts from page 1 of the website to page 30, I expected that adverts from the beginning of the dataset could be of better quality (i.e. attrative prices, more pictures or more attractive features). Therefore, all adverts were first shuffled before segregating them into training and validation sets. The training dataset represents around 1900 paired pictures-features which leaves about 700 paired pictures-features for validation. As I consider these numbers to be still quite low to train deep learning algorithms, no test dataset was built to keep more data for training and validation. Finally, all features were normalized to match a scale between -1 and 1. This is an essential transformation to allow the chosen optimizer to reach a minimum loss when running a graph session.

Building the Graph

Building a computational graph for tensorflow is one of the key step to obtain good results. It is also a particulary difficult task and needs to compute sessions many times to observe what layouts are better than others to obtain good validation accuracy. In that purpose, I tried a range of different combinations and evaluated their impact on results. However, it will be noticeable that the tested approaches are far from being a complete overview of potential techniques to improve the validation score and that my calculation of accuracy is not adequate to observe small size improvements over runs with a large amount of steps.

One way to potentially improve validation is to train a logistic model instead of a linear model. This can be done by creating classes of prices regrouping many adverts within a similar range of prices. Therefore, we also greatly reduce the number of prediction labels needed to run the graph. To perform this technique, I created ten classes of prices based on quartiles segregation so each classe has a similar number of adverts in it. This particularity permits that the model is trained on the same amount of information regardless of the price classes. Another well known technique in tensorflow is to increase the depth of the graph by introducing hidden layers with RELU activation in between inputs and outputs. This permits the model to detect finer information as the pictures will go through each layer. Lastly, I tried to implement parallel neurons on the same hidden layer that will only process either pictures or house features. The resulting tensors are then merged before going through the output layer.

Tuning the Graph
Tuning the model to perform better is another hot topic in deep learning. In the range of techniques that I could use, I triend to change from gradient descent optimizer to other types of optimizers. At the end, I kept adgrad optimizer to carry on analysing but the decrease in loss was not significant enough to justify such a choice. Changing the size of the hidden layer is also a possible way of improvement but it is also known to considerably increase running sessions times when it becomes bigger. Therefore, only small scale changes were tried which did not led to significant improvement. A technique known as the learning rate decay was also tested which involve reducing progressively the learning rate when the loss stay stable for long enough that a small decrease in learning rate can permit an improvement. However, this also means that the loss will decrease slower and slower which increases run times. With a very unstable training accuracy, I preferred to set quite a low learning rate from the beginning. To decrease run times, a good trick is to feed the graph with mini-batches instead of the full training set. However, the integrality of the validation set is kept when evaluating the model score. The size of the mini-batches can influence the score and needs to be assessed. For my computation needs, the mini-batches were set to a size of 40 which kept similar results than without the batch approach.

Results
The first graph tested included only pictures to predict prices and resulted in a maximum of 31.2% of accuracy in training and 28.6% 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. Adding the supplementary pictures from shuffling pixels resulted in accuracies quite close to the linear regression model without this operation : 31.0% accuracy for training and 29.4% for validation After building a logistic regression graph, the accuracy sharply increased over the training dataset with a maximum of 93.6 % bu the validation accuracy dropped to 20.1%. Therefore this model cannot predict accurately prices when pictures outside from training are used. To evaluate the potential of including house features in the neural network, I first built a graph with only these features included and the results show a maximum accuracy of 45.2% on training and 13.1% on validation. Still validation fails to be satisfactory but it seems that some information can be extracted. 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. However, when trying two hidden layers, it did not improve these results : accuracy of 65.9% for training and 19.0% for validation. When having both pictures and images on the same graph in two different layers, I

could reach ... for training and .... for validation. However, these results necessited a huge amount of steps : .... Finally, I tried to have pictures and images on the same layer but with two different neurons which led to 93.2% for training accuracy and 88.6% for validation with 9 millions steps. However, as said before, the way I calculated the accuracy values does not constitute a good indicator of improvement. It was more obvious when comparing predictions and real prices on a graphic that this method was providing better results than with the others.

Conclusion

I tried to improve the model by different means but it is obvious that using neural networks over such a small amount of data is not adequate. However, I am quite happy of having learn so much in the little time given to build this project. Deep learning becomes a hot trend and applications are becoming easier to develop by enthousiasts. I also see the potential from building more complex computational graphes that seem to improve the results even if they need long run times and a lot of forward thinking.

Going Further

I can think of two axis of improvement : Try convolutions over the pictures to evaluate if more information can be extracted to predict prices. The second axis is to build a better graph with a more complex architecture of hidden layers. More data et better computation run time by using gpu instead of cpu calculation would also be a great improvement.

How to run the project

First download the data from the dropbox link sent on Slack. Then, it is still not in user-friendly shape but the logical order to run the scripts would be the following :
imageTransformation.py : get original images from source file to derived dataset and change them to wanted size.
jsonFileTransfert.py : bring json files from source to derived dataset and delete folders if no json present.
jsonHandler.py : extract the features and prices from the json files in derived dataset and save the results in a csv file.
TensorFlowHousingImages.ipynb : will create the dataset between training and validation with shuffle and pickle files.
all others jupyter notebooks that are analysis based. To ease examination, the csv file and the final pickle file are already included in the repertory.