Machine learning model for housing prices in Ames,Iowa

Group 12, 18.11.2022

SCOPE

The business objective of this project is to create a model that predicts housing prices of houses in Ames, Iowa.

We will offer our machine learning model to various businesses, organizations or people who you could use the predictions to consider if a house is worth buying/selling. We can assume that there are machine learning models already employed to solve this problem today, so our aim is to create a model that performs just as well or better than existing models.

Doing this task manually would involve collecting data about house's in the area and using the average/median price of houses in the same proximity with similar features. This way would be quite tedious and unlikely to produce valuable results. Because, while two houses could be very similar, small differences in features could result in significant price differences. To what degree the small difference in features results in a big price difference, is going to be hard to calculate manually.

The performance of the model will be measured by the profits, or lack of profits, generated thanks to the accuracy of the model.

Initially, the stakeholders of the project will be the developers of the model, as we would deploy it and charge a fee for predictions. If the model performs well, then there is the possibility of selling the model to another party. This party could be a company or any group of people, and in such a case they would become the stakeholders.

The proposed timeline of the project will be:

1-3 months:
Gather data for the model
3-5 months:
Develop and train the machine learning model Milestone 1
5-7 months:
Deploy the model Milestone 2
7-12 months:
Monitor, update and maintain the model

To develop a machine learning model, we will have to collect many different features of the house's that we're going to predict the price of. The variables that describe the houses are lot size, neighborhood, foundation, heating, garage size and many more.

We would require a database for storing housing information. We would also have to store several iterations of the model to measure its performance over time. High-performance computers could also be needed for computing the predictions. Personnel for monitoring the performance of the live model would also be necessary. The personnel would be responsible for feeding the model fresh data and preventing the model from degrading.

METRICS

We would say that deciding on how accurate this project must predict house prices for it to be a success is a little bit subjective. There must be a balance between speed, convenience and accuracy. An experienced house salesperson can perhaps go through a house and predict a price that is almost 100 % accurate, but that is a "time-consuming" process compared to plotting data on a computer. So we ultimately set a goal of predicting prices within 5-10 % of the correct price.

To determine whether the solution is working we will use scoring and error metrics, like Mean squared error, Mean absolute error, R2 score and so on. They are though best used to compare different solutions and selecting the best one.

DATA

The available data consist of a dataset with around 3000 entries where they are split 50/50 into a training and test set. They both have almost 79 features or parameters. These features vary from lot size in square feet to what number of kitchens are available. If we go with the 10 times rule, we will need minimum 79 * 10 = 790 entries to be able to work with the data, which we certainly have. The way we would obtain more data would be having it deployed and used by real word applicants, which for this project is not feasible.

The training data will contain the ground truth label in Saleprice. We will construct models and train them on the training data with the Saleprice label, and later use the test set for prediction. The model will be based on this prediction. We will ensure accurate labels by scoring the models that are using the training data to make sure that they are accurate enough.

There will be a need for data cleaning as there are different types of values, and possibly scaled. This will be easier to see in the notebook where we can dive deeper into the data.

MODELING

After taking a look at sklearns “choose the right estimator” we can assume that models we will explore are different linear models like RidgeRegression, Lasso and SVR, and ensemble models like KNeighboor and Random forests. We will estimate performance based on scoring metrics like mentioned before, but also by predicting on the test set and submitting a submission to the kaggle competition. This can be an okey estimator on how the model is performing compared to other solutions. There will also be a need to check for over or underfitting. After deciding a model, we will hypertune it using some sort of grid search. When hypertuning is done we can use the grid search built in functions like best parameters with function importance to detect what features are most important for this particular model. And then we can use that information to do changes in the data cleaning section, like removing a column that has no impact.

DEPLOYMENT

The model(s) will be deployed using the flask framework. In an ideal world we would save predictions made with the model, as long as they are valid inputs, and use them in the model to hopefully get better performance. And after some research, it seems that flask has a built-in functionality where you can monitor the application. However, our knowledge in Flask is not particularly good, so if we can get a local application up and running we might be satisfied with that.

REFERENCES

List sources you’ve used during the planning of the project. The list of references should indicate the feasibility of your project.

https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

https://machinelearningmastery.com/much-training-data-required-machine-learning/

Summary (written after model and deployment completion, read last)

We wanted to explain some of the shortcomings and some sum up our experience working with this project. One major negative thing was that we started a bit too late on the deployment part which had a negative impact on how much work we did on

the model after deploying, which led to it not fully becoming a ML development cycle. It became difficult to do changes on the model after deploying because of not realizing that the way we constructed the model was not scalable (more specifically, we didn't have a good pipeline setup for processing the data).

Modelling:

We are generally satisfied with how the model turned out itself and the scores we were able to get on the kaggle competition. The data cleaning section we think turned out good and it was enjoyable to work with. We wish that we had spent more time on trying to create new features based on existing features. A bit more in depth explanations would be beneficial.

Deployment:

We ended up using the tutorial projects (hospital application) when building our own flask project for deployment. We spent too much time just getting it to work with our setup, and we did end up having a lot of features that had to be filled in (and alot of forms). We should have done more with feature importance, and maybe have a separate pipeline for the deployment that reduced the number of inputs (compared to the kaggle competition). In general we struggled when exporting models and pipelines from the notebook to the Flask project, in the future having everything in the same location would be the best. The XGBRegressor proved impossible to export, so we ended up basically copying the pipeline and XGBRegressor code into the Flask project. We did manage to export a random forest model, and could have of course used that instead. The consequence of all this was that doing changes to the model would require changes in both the Flask project and the notebook.