bostadspriser

Property price scraper and predictor from Hemnet

Introduction

This project is a web scraper for the Swedish property site Hemnet. It scrapes all the properties for sale in Sweden and stores them in a database. It also scrapes the sold properties and stores them in a database. The data is then used to train a machine learning model to predict the price of a property based on its features. The model is then used to predict the price of the properties for sale.

Gather data from Hemnet

Since Hemnet does not offer a public API, we have to scrape the data from the webpage. This came with a number of challenges, such as determining what has been scraped already and what has not. We solved this by storing the data in a MongoDB database in an iterative fashion. We started by finding location IDs that Hemnet uses and stored them in the database. Then, using these IDs, we constructed URLs that we could use to scrape the data. We then stored the data in the database. We then iterated over the database, finding the location IDs that had not been scraped yet, and repeated the process. For every search we did, we received up to 1000 listings. We then iterated over the listings, finding the ones that had not been scraped yet, and repeated the process.

This design is inherently asynchronous, which meant we could use a number of threads on a number of computers to scrape the data.

We used a total of 5 computers, each with 10-50 threads, to scrape the data. We scraped the data for around 3 weeks, and ended up with a total of roughly 1.3 million listings.

Furthermore, since the scraping is essentially done by mimicing a search in the webpage, we had to design clever ways to search for listings that have not been scraped yet.

The code for scraping the data is in the `scraping` folder. The code is written in Python.

Clean the data

The data we received from Hemnet was rather messy. There were a lot of inconsistencies in the data, such as missing values, incorrect values, and so on. We had to clean the data before we could use it for training the model.

Due to the inconsitencies, we had to compromise on the amount of data we could use and the quality of the data (which is reflected in the quality of the model's performance). We ended up with around 1 000 000 listings that we could use for training the model.

The code for cleaning the data is in the `cleaning` folder. The code is written in Python.

Prepare the data

We added an intermediate step before starting to train the model. This included one-hot encoding the housing form, for instance `housingForm="Lägenhet"` or `housingForm="Villa"` became `isApartment=true/false` and `isHouse=true/false` respectively. This was also done for `housingCooperative`, which became `hasHousingCooperative`. This was deemded necessary since the model would not be able to interpret the string values, and there were far too many possible values to use one-hot encoding.

We also converted date fields, such as `constructionYear` and `soldDate`, to the number of years since the year 2000 to improve the performance of the model.

The code for preparing the data is in `model/preapre.py`.

Train the model

Our goal with training the model was to try several different regression models and compare their performance. We used the following models:

- Linear regression
- Ridge regression
- Lasso regression
- Random forest regression
- Gradient boosting regression

Multiple models

We also aimed to create versions of the models, depending on when they were trained, which was done by adding timestamps to the model. But we took this a step futher since, depending on the parameters received in the API, differently trained models might be needed.

For example, when training a model with **asking price** (intial price set by the seller) as a feature, it gives entirely different results than when training a model without the asking price as a feature. Therefore, if a user would like to know "What should this cost based only on its properties?", it would use a model trained without asking price. If a user would like to know "What price will this end up with?", it would use a model trained with asking price.

Hyperparameter tuning

We used a grid search to find the best hyperparameters for each model. We used the following hyperparameters:

- Linear regression: None
- Ridge regression: alpha
- Lasso regression: alpha
- Random forest regression: n_estimators, max_depth, min_samples_split, min_samples_leaf
- Gradient boosting regression: n_estimators, max_depth, min_samples_split, min_samples_leaf, learning_rate

Use the model

We created a website and a REST API to demonstrate the model. The website is available at bostadspriser.app.cloud.cbh.kth.se and the API is available at bostadspriser-api.app.cloud.cbh.kth.se.

The website allows the user to see the predictions of a list of live and active property listings from Hemnet. The user can also input a custom property listing and see the prediction for that property, which is useful when the user is interested in selling their property.

The code for the website is in the `frontend` and `api` folders. The code is written in JavaScript and Python respectively.

Transparency

We aimed for transparency in the model, which is why we included a page presenting all the features user in the model, as well as the model type (such as linear regression or random forest regression) and the hyperparameters used for that model.