Work report, House prices prediction

Author: Tomer Back

Goal: build a model that predicts the price of houses in Utah according to 79 known features.

**Preprocess:**

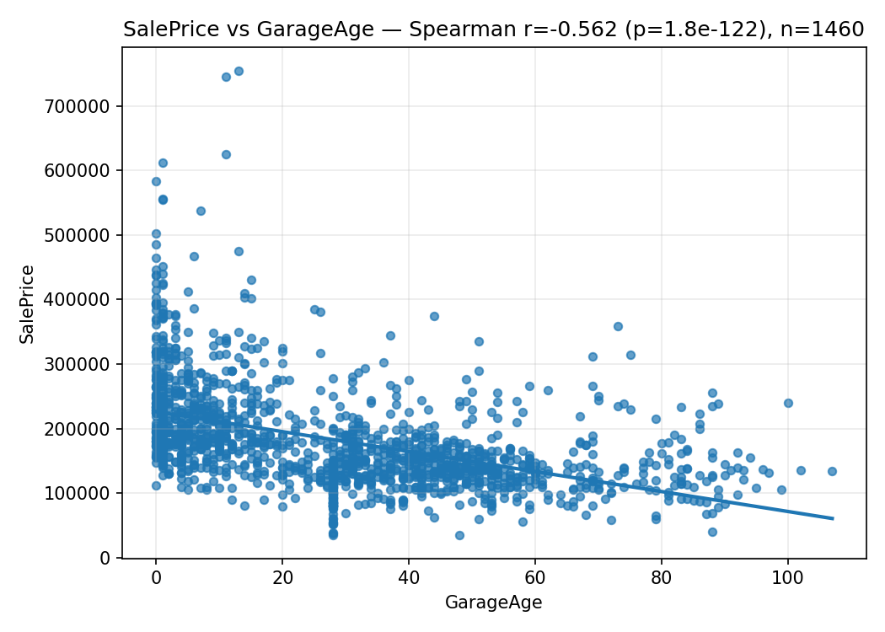
A lot of the data needs some processing before being trained on.  
the first thing to do is to identify features that are not given as numbers (that includes features that the numbers are just tags and don’t have a natural order like MSSubClass).

Ones identified we need to address two types of features:

1. **Features with natural order** (for example ExterQual that can get values like excellent, good, etc.). Ones identified we can give numbers to each possible feature inputs (so for the earlier example we shall get 5 for excellent, 4 for good, etc.).  
   **small thing to watch out**: We don’t know the effect of such features on the target column (the price) and we don’t necessarily know what was the original intention , so giving such "linear" numbers does not necessarily represent the data correctly.  
   For example, It could be that ExterQual - Excellent is something only extraordinary houses get so the difference between good and excellent is very big but the difference between good and average is not such drama. That means we gave a linear connection for something not linear at all! Doing such thing can limit later our model selection.
2. **Features without any natural order** – something like the neighborhood or the shape of the lot. For those features I used one-hot encoding, meaning for each possible input i added a column that gets 0/1 according to the data in the original feature. For example if we have 3 lot shapes: square, rec and circle. I added 3 columns/features lot\_square, lot\_rec, lot\_circle.  
   This is not always the best solution, for features with a lot of options it's not always the most viable solution for couple of reasons (for reasons I won't mention), but I found it good enough here (the main concern was neighborhoods feature, but after a bit of graph making, I figured that variance of each neighborhood is small which makes it a strong feature especially for RMSE ranking. (as a system this is a wrong thing to do, and this will hurt the credibility of my own model evaluation because I already made decisions looking at the test, saying that, I would have come to the same conclusion looking even at a small part of the train so even though I did this part looking at all of the data its fine by me for this task)

From here there are more generic but important preprocessing to do.

* תמונה שמכילה טקסט, צילום מסך, קו, עלילה

  תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.Cut low variance columns, an example here would be Utilities which are almost completely constant except one example. Keeping it can overfit our model.
* Something similar happens with new one-hot columns, there we need to be careful not to delete any columns from the train and not from the test (or the opposite).
* תמונה שמכילה טקסט, צילום מסך, עלילה, קו

  תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.Feature engineering: this is basically adding age features like a house age, is renovated(?) etc....  
  as seen in graphs below, such features are highly correlated to the price. We are adding such features because this is a natural relation between 2 other features we could highlight without the use of a model.

**EDA:**

There are many ways to look at the data and extract insights. The first thing I did was to plot each feature vs price and calculate the correlation. This gives some good ideas on what the most important features are. This does not capture complex relations that contains more then one feature, an example for such relation could be garage size and pool size, we expect houses with pools to have less space for a garage, but not of homes have pools, so from data we can make generalization that "bigger garage -> expensive house" but out theory states that houses with pools will have less garage space, but still they will be more expensive because of the pools.

Saying that due to the fact we are predicting the price of houses, it's not a surprise that some features give us a good idea about the order of magnitude of a house price, so features like number of rooms, lot size, amount of baths, garage size and other simple obvious features could indicate for us what is the general price of a house.

Looking at the list of correlations between the prices and the features shows that in not such a bad way:

|  |
| --- |
| Feature, Correlation to price |
| OverallQual, 0.809829 |
| GrLivArea, 0.731310 |
| GarageCars, 0.690711 |
| ExterQual, 0.684014 |
| BsmtQual, 0.678026 |
| KitchenQual, 0.672849 |
| YearBuilt, 0.652682 |
| HouseAge, -0.650120 |
| GarageArea, 0.649379 |
| FullBath, 0.635957 |
| GarageFinish, 0.633974 |
| TotalBsmtSF, 0.602725 |
| SinceRemod, -0.576582 |
| 1stFlrSF, 0.575408 |
| YearRemodAdd, 0.571159 |
| GarageYrBlt, 0.563256 |
| Foundation\_\_PConc, 0.562287 |
| GarageAge, -0.562258 |
| FireplaceQu, 0.537602 |
| TotRmsAbvGrd, 0.532586 |
| Fireplaces, 0.519247 |
| HeatingQC, 0.491392 |
| OpenPorchSF, 0.477561 |
| LotArea, 0.456461 |
| GarageType\_\_Attchd, 0.455399 |
| GarageType\_\_Detchd, -0.440294 |
| MSSubClass\_\_60, 0.436418 |
| MasVnrArea, 0.415906 |
| MasVnrType\_\_None, -0.384642 |
| HasVeneer, 0.384642 |
| MSZoning\_\_RM, -0.380777 |
| LotFrontage, 0.375590 |
| Ext\_VinylSd, 0.369873 |
| Foundation\_\_CBlock, -0.368044 |
| BsmtFinType1, 0.361625 |

So only small amount of have a strong (independent) correlation to the price.

We can capitalize on that by building a simple model that’s trained on the principle of finding the order of magnitude of a house and will not care for small differences, such model will come in handy later. Using a relatively small decision tree based on polynomial loss (degree>2) I was able to get relatively good results, more on that later.

Another important thing to understand about the data is its small size. We only have 1500~ rows of data to play around with. This is not a lot and can make the task of finding patterns hard, especially with so many features.

תמונה שמכילה טקסט, צילום מסך, קו, עלילה

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.My main concern is overfitting, due to the small size of the data we can find patterns where there aren’t any, for example we can see that all houses with asphalt exterior are relatively cheap, but those are just 3 houses!

That’s not enough to infer any patterns from such data, and although we tried deleting small variance features that won't necessarily because from the start, we don’t have a lot of data, especially when we use some of the training data provided to test.

So, what do we do? A complex model that considers all small features can infer such wrong correlation, on the other hand evaluating the price of a house is a difficult task that requires a model to consider all the features and even find some not trivial connections between different features (one such feature can be if a house has a pool or not).

This understanding brought me to the following solution.

**Model**

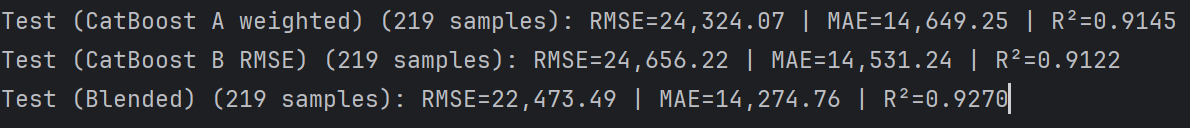
So, we need a model that is simple on one hand but complex on the other hand.

My idea was to use two models, one simple with the goal of finding an estimate to the price reliably (meaning we try to use minimal number of features that are common between all houses and give a good estimate of the price, such feature can be the number of rooms in the house). The second model will be a complex model designed for such complex tasks (more on the specifics of the model later).

Most of the time we will use the complex model guess, but assuming the reliability of the simple model of finding the general price, if we find a big difference between the two, we will use the simple one.

Unfortunately, I couldn’t make this idea better than the complex model I already used inside (standing as a solo model), best case senrio my new model was able to keep similar results. Although the unfortunate results I believe my approach could work with some more advanced techniques.

The best improvement I was able to make from the main model was to use two complex model training one on a different lost function, one that punishes more severely for large mistakes, after which combining some linear combination of the two models outputs.

The improvement came in my opinion mostly from the use of ensemble (meaning the use of more than one trained model) which reduces the possible error. It was interesting to see that the model trained on the different loss function (no RMSE) got a weight of 0.45 compared to 0.6 of the second model when optimized (there is some bias), maybe showing some sparks that my previous idea could work.

As for models used, for the complex model I ought to use CatBoostRegressor, and for a simple one a simple decision tree.