Problem

Client

Data

Exploratory Data Analysis

Inferential Statistical Analysis

Modeling

Conclusion

References

# Modeling

## Simple linear regression

We start with simple linear regression model to use it as a benchmark to compare other more complicated and less interpretable models against it. First, we split the data into train and test set where training set contains 75% of the observations, which is 20127 and test set contains 6710 data points, which is 25% of the observations. Next, we fit simple linear regression model with an intercept term via statsmodels OLS.

Table 1. OLS Regression Results

|  |  |  |  |
| --- | --- | --- | --- |
| **Dep. Variable:** | Time Adjust Sales Price | **R-squared:** | 0.769 |
| **Model:** | OLS | **Adj. R-squared:** | 0.768 |
| **Method:** | Least Squares | **F-statistic:** | 1450. |
| **Durbin-Watson:** | 1.993 | **Prob (F-statistic):** | 0.00 |

The model’s R-squared is 0.769. The test set R-squared is 0.769, which we set as our benchmark R-squared. The model doesn’t overfit. t-statistics look good for all coefficients except “Design\_Det 1 Sty Condo” and “Market Area\_105”. The Durbin-Watson is right around 2 and between 1.5 and 2.5, so autocorrelation is likely not a cause for concern.

## Principal component analysis

Since many features in our dataset tend to be correlated and to explore the data further, we perform Principal Component Analysis to see if variation could potentially be captured by a few features and potentially reduce overfitting.

Chart, bar chart, histogram

Description automatically generated

We can see that even the largest component only explains about 9% of the variation, with the next 5% of the variation and then slow decline with plateau at about 3% for every additional component. Given this result, it doesn’t make sense to use the PCA for our problem since the variance that’s captured doesn’t tend to be in a few major directions, but instead distributed across features.

## XGBoost

XGBoost is one of the most popular machine learning frameworks among data scientists. According to the Kaggle [State of Data Science Survey 2021](https://www.kaggle.com/kaggle-survey-2021), almost 50% of respondents said they used XGBoost, ranking below only TensorFlow and Sklearn[[1]](#footnote-1).

XGBoost is an optimized distributed gradient boosting library. It implements machine learning algorithms under the [Gradient Boosting](https://en.wikipedia.org/wiki/Gradient_boosting) framework. XGBoost provides a parallel tree boosting that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples[[2]](#footnote-2).

Here we want to solve our regression problem. As with many other ML algorithms, there is an issue of parameter tuning[[3]](#footnote-3).

Here we focus on 3 main parameters:

* learning\_rate- eta shrinks the feature weights to make the boosting process more conservative. Default 0.3 [0,0.2,0.5]
* min\_split\_loss Gamma specifies the minimum loss reduction required to make a split. Default 0 [0,10,100]
* reg\_lambda – default 1, L2 regularization term on weights (analogous to Ridge regression). Increasing this value will make model more conservative. 1,10,100

The results show that best eta is 0.5 and other parameters are good at their default values.

Best model’s cross-validation RMSE is 166,008 USD. After training model on a full training set the RMSE is 150982.546 with corresponding R-squared 0.89.

Let’s explore XGBoost’s feature importance table.

Table

Description automatically generated

Discuss the graph.

Chart, scatter chart

Description automatically generated

Discuss the fit.

## Random forest regression

We use Random Forest Regressor from sklearn.ensemble. A random forest is a meta estimator that fits several decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting[[4]](#footnote-4).

We first use the default model on the training data. The training data R^2 is 0.98, the test data R^2 is 0.91. This tells us that there’s some overfitting going on.

Perhaps tuning parameters could help with that.

The default model has n\_estimators=100 (number of trees in the forest), uses squared error as a criterion, has no max\_depth of the tree, The minimum number of samples required to split an internal node is 2 (**min\_samples\_split=2**), The minimum number of samples required to be at a leaf node (min\_samples\_leaf=1), The number of features to consider when looking for the best split, default max\_features=n\_features.

Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree. (default =True)

Since default parameters lead to fully grown unpruned trees and likely lead to overfitting, as we have seen, we want to search parameter values that are more restrictive.

We set up the optimization problem[[5]](#footnote-5) we choose to search between 10 and 100 trees, consider either all or only 30% of the features, check max number of levels per tree between 10 and 110, allow for 2,5 or 10 Minimum number of samples required to split a node, allow for 1,2,4 Minimum number of samples required at each leaf node, use or not use bootstrap.

We perform Random search of parameters, using 3-fold cross validation, Fitting 3 folds for each of 50 candidates, totaling 150 fits

The resulting optimal model is below:

{'n\_estimators': 30,

'min\_samples\_split': 5,

'min\_samples\_leaf': 1,

'max\_features': 0.3,

'max\_depth': 60,

'bootstrap': False}

Model Performance

Average Error: 66121.3632 degrees.

Accuracy = 91.14%.

Model Performance

Average Error: 61498.4644 degrees.

Accuracy = 91.60%.

Improvement of 0.50%.

So, according to our hyperparameter optimization, we can gain 0.5% in accuracy improvement. I think we could use either base model or optimized model, but the selected model is much more restrictive yet yields similar performance. Let's use selected optimal model!

Let’s see the 6-fold cross validation on test set yields Mean R^2: 0.8922873538889678

Standard deviation R^2: 0.013079204638458948, there is some variation between the folds, which tells that algorithm is still a little sensitive to splits.

Final optimized model evaluation yielded 0.912 R2. RMSE is 127313.041.

Chart, scatter chart

Description automatically generated

It looks like random forest works well for houses under 1,000,000 USD, however as price increases, we see the performance decreasing. Some of the more expensive houses over 3 million are undervalued, however that can just be the case and there are not many of them. Interestingly, there’ s more severe undervaluing by the model as opposed to overvaluing.

## Polynomial regression

Next, we obtain polynomial features of second degree, including interactions, which expands our feature set to 1225 and we fit simple linear regression on this new set of features.

Training set R2 is 0.9113435115233615, however test set R2 is 0.659 which shows severe overfitting.

Chart, scatter chart

Description automatically generated

Looks beautiful, a bit of issues for high sale price properties, but doesn’t seem to over/underestimate systematically.

## Linear regression with log response

For an input with a larger amount of relative variation (for example, heights of children, or weights of animals), it would make sense to work with its logarithm immediately, both as an aid in interpretation and likely an improvement in fit too.[[6]](#footnote-6)

A multiplicative model on the original scale corresponds to an additive model on the log scale. For example, a treatment that increases prices by 2%, rather than a treatment that increases prices by $20. The log transformation is particularly relevant when the data vary a lot on the relative scale. Increasing prices by 2% has a much different dollar effect for a $10 item than a $1000 item.

(reword it)

Test and train set R^2 are equivalent at 0.894 RMSE looks pretty large :343824.09 USD.

So perhaps we’ll omit this model.

1. https://www.datacamp.com/tutorial/xgboost-in-python [↑](#footnote-ref-1)
2. https://xgboost.readthedocs.io/en/stable/ [↑](#footnote-ref-2)
3. https://www.kaggle.com/code/prashant111/a-guide-on-xgboost-hyperparameters-tuning/notebook [↑](#footnote-ref-3)
4. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html [↑](#footnote-ref-4)
5. https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74 [↑](#footnote-ref-5)
6. https://statmodeling.stat.columbia.edu/2019/08/21/you-should-usually-log-transform-your-positive-data/ [↑](#footnote-ref-6)