Predicting House Prices

Adam Maghout

2025-01-19

Introduction

Background

As house prices skyrocket due to inflation and overpopulation in urban areas (Goda et al., 2020), keeping them in a check is a priority for local governments (Dawkins & Nelson, 2002). Conversely, from a landowner's perspective, the recent surge in demand carries with it the promise of making large returns on property investments (Bano, 2024). Knowing what increases the value of a property thus constitutes an important discussion point for government bodies and private investors alike. Is the construction of a pool going to increase the value significantly more than expanding a house into its garden for instance? Often what customers and renters look for most is not reflected in the price of a property, so the driving force behind prices is hidden to landowners (Tuzovic, 2009). Furthermore, as the value fluctuates with time, it is crucial to devise models to predict how assets will affect property valuations.

Several machine learning (ML) techniques have been applied in the field of real estate. These include random forests (RF), support vector machines (SVM) and gradient boosting machine (GBM). Ho et al. (2021), for instance, found RF and GBM models to perform better at predicting prices over time, whilst SVM models produced reasonably accurate predictions within a tight time constraint. Calainho et al. (2024), provided further support for the use of ML techniques as they performed better than linear models in their study investigating commercial real estate transactions in New York.

The current research is done as part of a Kaggle competition. The competition makes use of the Ames Housing Dataset, which was first presented in 2011 as an alternative to the Boston Housing Dataset, a then-popular dataset for training models in the field of machine learning and data analysis (De Cock, 2011). It contains various features and attributes of residential homes in Ames, Iowa, USA. In line with other studies cited above, the objective of this research is to present an overview of various ML techniques for predicting property value.

Problem

This project aims to predict house prices for a test dataset using various variables relating to the size of the house, its type as well as other property-related variables. The target variable is `SalePrice`, corresponding to the value of the house in dollars if it were to be sold. Performance is measured using the Root-Mean-Squared-Error (RMSE) between the logarithm of the predicted value and the logarithm of the observed sales price. The aim is to minimise the RMSE, indicating a good prediction model has been reached using the available variables. This is in correspondance with the rules set out by the Kaggle competition. Additionally, researchers are not given access to the test data and can only evaluate the performance of their model by submitting the resulting predicted property prices through the Kaggle website, after which the RMSE on the test data is given and a ranking is assigned to the researcher's submission. As the number of permitted submissions is not limited, it is also entirely possible to approach the problem without a model and just adjust predictions after each submission until the RMSE is lowered (although this is cumbersome).

Data Description

Descriptives of the variables are given below. An in-depth explanation of each variable's substantive meaning and levels is given in the Data folder, which reveals that variables pertain to the location and internal components of a house. Several variables are continuous, such as `1stFlrSF`, the First Floor square footage, whilst others are categorical, such as `Heating`, the type of heating. Both types of data can be used for conducting a regression analysis so this is of no worry for the present study.

Furthermore, several variables are subjectivelly measured, such as `BsmtCond`, the quality of the basement, for which no information is given concerning the appraisal process. For the purposes of the current research, however, we consider the effect of subjectivity to be negligeable on obtaining a viable model.

```
Number of variables:[1] 81
Number of observations[1] 1460
[1] "Range of prices observed: 34900 to 755000"
```

Data Cleaning

Before running any models, an adjusted version of the data is obtained to ensure results are correct. The categorical variables in the dataset need to be converted to factors so as to be handled correctly by the ML models. Furthermore, some of the variables are total variables, that is, they are the sum of two or more other variables. This is the case for `TotalBsmtSF` and `GrLivArea` which are respectively the aggregated totals of the basement and general living area square footage. We remove them from the dataset used for prediction, as well as the house `Id` which cannot be used for prediction. No other variables are perfectly correlated with each other, yielding a total number of predictive variables equal to $81 - 3 - 1 = 77$ (since `SalePrice` is the outcome variable).

Missing Data

Rationale

Given the presence of missing data in both the training and test datasets, identifying an imputation method that works on the training data should also improve predictions on the test data. Many variables in the training dataset have missing values. These are given by:

```
                Variable MissingCount
PoolQC             PoolQC         1453
MiscFeature   MiscFeature         1406
Alley               Alley         1369
Fence               Fence         1179
FireplaceQu   FireplaceQu          690
LotFrontage   LotFrontage          259
GarageType     GarageType           81
GarageYrBlt   GarageYrBlt           81
GarageFinish GarageFinish           81
GarageQual     GarageQual           81
GarageCond     GarageCond           81
BsmtExposure BsmtExposure           38
BsmtFinType2 BsmtFinType2           38
BsmtQual         BsmtQual           37
BsmtCond         BsmtCond           37
```

```
BsmtFinType1 BsmtFinType1           37
MasVnrType     MasVnrType            8
MasVnrArea     MasVnrArea            8
Electrical     Electrical            1
```

Variables with missing data appear to represent features that do not apply to all houses. For instance, `GarageType` (the type of garage) and `GarageYrBlt` (the year of construction of the garage) are likely missing for houses without garages, whilst `PoolQC` (pool quality) can obviously not be measured for houses without a pool. As such, it can be assumed that no real missing data is present, which is a reasonable assumption given the data is provided in the context of a programming competition.

Dealing with Missing Data

We replace missing values with 0 and "none" for numeric and categorical variables respectively. This appears to be consistent with the missing data pattern observed in the previous section, as the absence of a feature signifies it has no area and can thus not add value to a house. The imputation by these new values is done in a separate dataset to enable measuring the impact of the imputation procedure on the results.

Fitting a Model

Models

We evaluate several models to predict `SalePrice`:

- Linear Regression
- Quadratic Regression
- Random Forests
- Support Vector Machines (SVM)

Models are first ran with generic parameters in a first round. In a second round, only promising candidate models are retained, for which the parameters are tuned. This has the disadvantage of potentially discarding models that would perform well with other settings. However, given the quantity of models considered, the parameter space is too large to visit for each model so this simplified two-step approach ensures the computational time is kept at a minimum. Furthermore, all models in R are constructed with default settings that can generally be assumed to have been chosen optimally by package developers.

Interactions between models, such as between a linear and cubic regression term, are left out in the first step, and only included in the second step if there is reasonable belief that this will

improve prediction. Given that the models need to be run twice, on both the dataset with and without missing data, there are thus 8 models that need to be run in step 1. To ensure further stability and applicability of the results to the test data, cross-validation (CV) is used, with the number of folds varying from 1 to 10. The decision to only fine-tune parameters in a second step is thus largely justified. As household values are unlikely to be negative, negative predicted values are replaced for all models by a conservative lower bound of 10 000$. To facilitate running the models, a function is created and ran several times. The details of how each model is ran are provided here but the entire function can be found withing the Scripts folder.

```
if (method == 'linear') {
        model <- lm(predictors, data = trainset)
        testset$Predicted <- pmax(predict(model, newdata = testset), 10000)

} else if (method == 'quadratic') {
        model <- lm(predictors, data = trainset)
        testset$Predicted <- pmax(predict(model, newdata = testset), 10000)

} else if (method == 'rf') {
        model <- randomForest(predictors, data = trainset_rf, ntree = 10000)
        testset$Predicted <- pmax(predict(model, newdata = testset_rf), 10)

} else if (method == 'svm') {
          model <- svm(predictors, data = trainset,
                       kernel = "radial",
                       type = "eps-regression")
          testset$Predicted <- pmax(predict(model, newdata = testset), 10000)
}
```

Intricacies of Methods

A hurdle when performing cross-validation on a dataset with a significant quantity of missing data is that often folds will not have the same number of levels for a categorical variable as in the whole dataset. This is an issue as levels that are present in the testing phase of a CV but not in the training phase will then posess no predictive attributes. This can be mitigated by combining levels in both the CV training and test sets (not to be confused with the overall training and test sets) such that they match for both groups. This reduces the precision of our models but is unfortunately necessary. The details of this procedure are given here.

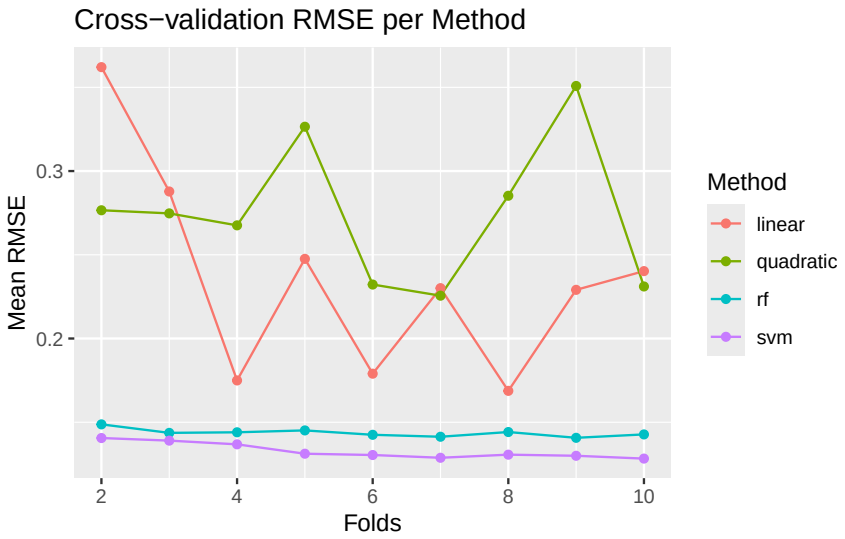
```
factor_vars <- trainset %>% select(where(is.factor))
for (var in names(factor_vars)) {
  # Get counts for each level in the training set
```

```
  trainset[[var]] <- droplevels(trainset[[var]])
  level_counts <- table(trainset[[var]])

  # Identify levels with fewer than 2 observations
  low_freq_levels <- names(level_counts[level_counts < 2])

  # If we have low-frequency levels, combine them
  if (length(low_freq_levels) > 0) {
    # Create a combined "Other" category with descriptive name
    combined_name <- paste(low_freq_levels, collapse = "_")
    trainset[[var]] <- factor(ifelse(trainset[[var]] %in%
                                       low_freq_levels, combined_name,
                                     as.character(trainset[[var]])))

    # Recount levels after initial collapse
    level_counts <- table(trainset[[var]])

    # If the combined level still has only 1 observation,
    # try to merge with the next least common level
    if (level_counts[combined_name] < 2) {

      # Find the next least common level,
      # only if there's more than one unique level
      sorted_levels <- names(sort(level_counts))
      if (length(sorted_levels) > 1) {
        next_level <- sorted_levels[2]

        # Update the combined level to be merged with the next level
        new_combined_name <- paste(combined_name, next_level, sep = "_")
        trainset[[var]] <- factor(ifelse(trainset[[var]] == combined_name |
                                           trainset[[var]] == next_level,
                                         new_combined_name,
                                         as.character(trainset[[var]])))
      }
    }

    # Apply the same transformation to the test set
    testset[[var]] <- factor(ifelse(testset[[var]] %in%
                                      low_freq_levels | testset[[var]]
                                    == next_level, new_combined_name,
                                    as.character(testset[[var]])),
                             levels = c(levels(testset[[var]]),
```

```
                                          new_combined_name))
    testset[[var]] <- droplevels(testset[[var]])
  }

  # After collapsing, check if there are fewer than 2 unique levels
  if (length(unique(trainset[[var]])) < 2 |
      !all(levels(testset[[var]]) %in% levels(trainset[[var]]))) {
    trainset <- trainset %>% select(-all_of(var))
    testset <- testset %>% select(-all_of(var))
  }
}
```

Performance during Cross-validation

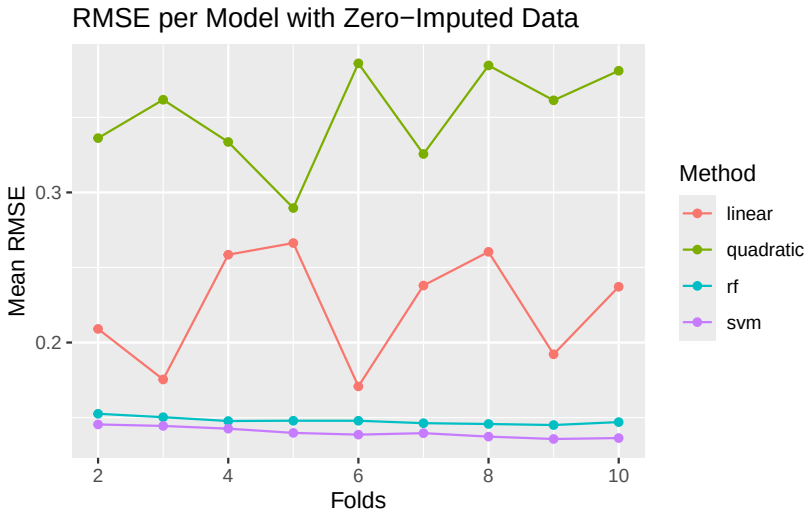
As specified above, we apply cross-validation with various fold counts (2 to 10) to evaluate the RMSE across the different model configurations.



Based on the graph, we select Random Forest and SVM as the best-performing models as they are the most stable across number of folds. We must however first compare performance with the model fitted on the whole data with imputed values.

Imputed Data Cross-validation

We run the cross-validation again, but this time with the missing data replaced.



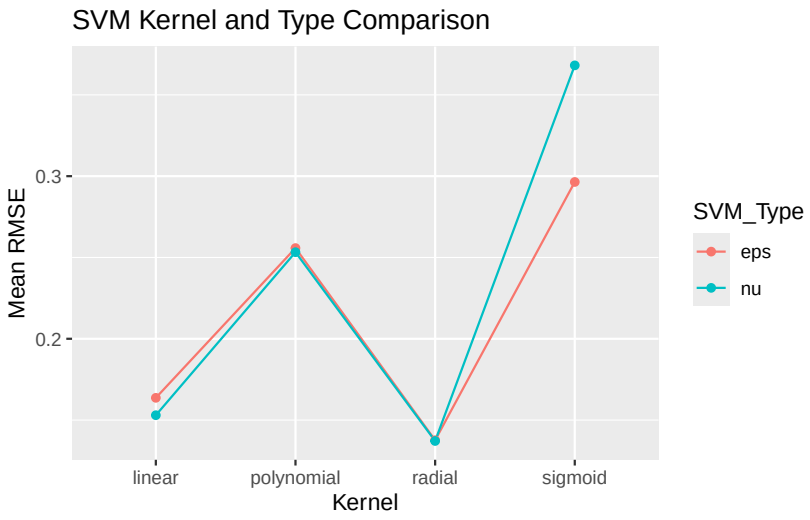
	Folds	Method	Mean_RMSE_no_missing	Mean_RMSE_zero_imputed
1	10	linear	0.2402436	0.2371552
2	10	quadratic	0.2310779	0.3810553
3	10	rf	0.1427250	0.1469969
4	10	svm	0.1282903	0.1363649
5	2	linear	0.3620604	0.2090750
6	2	quadratic	0.2765891	0.3361876

The difference in RMSE is minimal between the datasets with and without imputed data, indicating that the procedure defined earlier is justified. However, this may have been due to the number of variables with missing data being comparatively low. Nevertheless, SVM still appears to be the best model and can thus be perfected in the next step. As there is no obvious trend concerning whether the data should be imputed or not, the imputation procedure is retained as it makes substantive sense.

Choice of Parameters for SVM

We compare different SVM kernels (linear, polynomial, radial, sigmoid) and regression types (eps and nu) to find the optimal configuration. These are the only tunng parameters available for SVM models. For the previous step, the sigmoid kernel was used, alongside an eps regression so the aim is to ascertain whether this can be improved upon. 10-fold cross-validation is retained here for ensuring result stability.

```r
# SVM parameter options
svm_kernels <- c("linear", "polynomial", "radial", "sigmoid")
svm_types <- c("eps", "nu")
results_svm <- data.frame(Folds = integer(), Kernel = character(),
                          SVM_Type = character(), Mean_RMSE = numeric())

for (kernel in svm_kernels) {
  for (svm_type in svm_types) {

    # Run cross-validation (returns columns Method, Mean_RMSE)
    cv_result <- cross_validate_rmse(
      data = train_zero,
      target = "SalePrice",
      folds = 5,
      methods = c("svm"),
      svm_kernel = kernel,
      svm_type = svm_type
    )

    # Add columns for Kernel and SVM_Type
    cv_result$Kernel   <- kernel
    cv_result$SVM_Type <- svm_type

    # Return the results
    results_svm <- rbind(results_svm, cv_result)
  }
}

ggplot(results_svm, aes(x = Kernel, y = Mean_RMSE,
                        color = SVM_Type, group = SVM_Type)) +
  geom_line() +
  geom_point() +
  labs(title = "SVM Kernel and Type Comparison", x = "Kernel",
       y = "Mean RMSE")
```



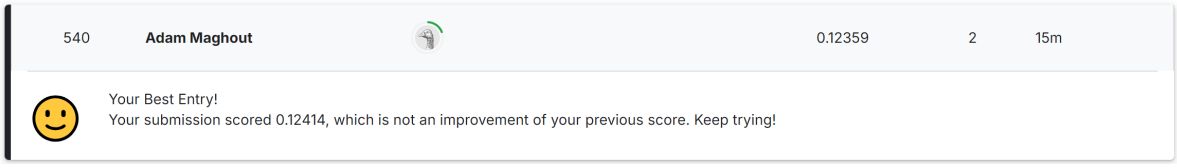
The radial kernel with nu-regression achieved the best performance, so we select this configuration for final testing, whilst retaining the imputation procedure.

Performance on the Test Set

Using the best SVM configuration, we generate predictions on the test set with imputed missing values (`test_zero`).

```
# Train final SVM model and predict on test_zero
svm_model <- svm(SalePrice ~ ., data = train_zero, kernel = "radial",
                 type = "nu-regression")
test_zero$Predicted_SalePrice <- predict(svm_model, newdata = test_zero)

# Prepare the submission file
submission <- data.frame(Id = test$Id,
                         SalePrice = test_zero$Predicted_SalePrice)
write.csv(submission, file = "Results/submission_Adam_Maghout.csv", row.names = FALSE)
```

This returns an RMSE of 0.12359 according to the Kaggle website, which positions the model in the 540th place on the leaderboard.



Improvements

The following considerations were made regarding model selection:

- The model could have been run several times.
- Cross-validation on the training data ensures the solution is not overly fitted but does not fully capture overall performance (even at 10 folds).
- Several model selection decisions are arbitrary, such as the lower bound selection, the number of folds and the original tuning parameters used.

Bibliography

Bano, N. (2024). *Against Landlords: How to Solve the Housing Crisis*. Verso Books.

Calainho, F. D., van de Minne, A. M., & Francke, M. K. (2024). A Machine Learning Approach to Price Indices: Applications in Commercial Real Estate. *The Journal of Real Estate Finance and Economics*, *68*(4), 624–653. https://doi.org/10.1007/s11146-022-09893-1

Dawkins, C. J., & Nelson, A. C. (2002). Urban containment policies and housing prices: An international comparison with implications for future research. *Land Use Policy*, *19*(1), 1–12. https://doi.org/10.1016/S0264-8377(01)00038-2

De Cock, D. (2011). Ames, Iowa: Alternative to the Boston Housing Data as an End of Semester Regression Project. *Journal of Statistics Education*, *19*(3). https://doi.org/10.1080/10691898.2011.11889627

Goda, T., Stewart, C., & Torres García, A. (2020). Absolute income inequality and rising house prices. *Socio-Economic Review*, *18*(4), 941–976. https://doi.org/10.1093/ser/mwz028

Ho, W. K. O., Tang, B.-S., & Wong, S. W. (2021). Predicting property prices with machine learning algorithms. *Journal of Property Research*, *38*(1), 48–70. https://doi.org/10.1080/09599916.2020.1832558

Tuzovic, S. (2009). Key determinants of real estate service quality among renters and buyers. *Journal of Services Marketing*, *23*(7), 496–507. https://doi.org/10.1108/08876040910995284