

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 called 'SalePrice', corresponding to the value of the house 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

```
# Load data
train <- read.csv('Data/train.csv')
test <- read.csv('Data/test.csv')

# Number of rows and column
cat('Number of variables:')
ncol(train)
cat('Number of observations')
nrow(train)
print(paste0('Range of prices observed: ', min(train$SalePrice),
            ' to ', max(train$SalePrice)))
```

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

Data Cleaning

Convert Characters to Factors

Many variables in this dataset are categorical, which we handle by converting character columns to factors for proper analysis and model fitting.

```
train <- train %>% mutate(across(where(is.character), as.factor))
test <- test %>% mutate(across(where(is.character), as.factor))
```

Handling Multicollinearity

Some of the variables are total variables, that is, they are the sum of two others. This is the case for variables TotalBsmtSF and GrLivArea. We remove them from the dataset used for prediction, as well as the ID column

```
train_clean <- train %>% select(-c(Id, BsmtUnfSF, GrLivArea))
test_clean <- test %>% select(-c(Id, BsmtUnfSF, GrLivArea))
```

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 dataset should also improve predictions on the test dataset.

Relevant Variables with Missing Data

Missing values in the training set:

	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 often represent features that do not apply to all houses. For instance, `GarageType` and `GarageYrBlt` are likely missing for houses without garages.

Dealing with Missing Data

We replace missing values with 0 (for numeric variables) or “none” (for categorical variables). This is done in separate datasets to measure the impact of the imputation procedure.

```
# Replace missing values in numeric columns with 0
# and in factor columns with 'none'
train_zero <- train_clean %>%
  mutate(across(where(is.numeric), ~ ifelse(is.na(.), 0, .))) %>%
  mutate(across(where(is.factor), ~ fct_na_value_to_level(., "none")))

test_zero <- test_clean %>%
  mutate(across(where(is.numeric), ~ ifelse(is.na(.), 0, .))) %>%
  mutate(across(where(is.factor), ~ fct_na_value_to_level(., "none")))

# Check for missing data to confirm replacement
missing_data_train <- sapply(train_zero, function(x) sum(is.na(x)))
```

```
missing_data_test <- sapply(test_zero, function(x) sum(is.na(x)))  
any(missing_data_train != 0)
```

```
[1] FALSE
```

```
any(missing_data_test != 0)
```

```
[1] FALSE
```

Fitting a Model

Options Formulated

We evaluate several models to predict `SalePrice`:

- Linear Regression
- Quadratic Regression
- Cubic Regression
- Decision Tree Learning
- Random Forests
- Support Vector Machines (SVM)
- Neural Networks (NN)

Models are first compared with general parameters and then the parameters for candidate models are explored further.

Intricacies of Methods

A lower bound of 10 000 is given to avoid issues with the RMSE of the likelihood.

```
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 == 'cubic') {
```

```

model <- lm(predictors, data = trainset)
testset$Predicted <- pmax(predict(model, newdata = testset), 10000)

} else if (method == 'tree') {
  model <- rpart(predictors, data = trainset, method = "anova")
  testset$Predicted <- pmax(predict(model, newdata = testset), 10000)

} else if (method == 'rf') {
  # Keep only variables with exact level matches
  factor_vars_rf <- names(trainset)[sapply(trainset, is.factor)]
  trainset_rf <- trainset # Avoid overwriting the data for other methods
  testset_rf <- testset

  # Identify variables to drop if levels do not match exactly
  vars_to_drop <- c()

  for (var in factor_vars_rf) {
    # Check if the levels match exactly
    if (!identical(levels(trainset[[var]]), levels(testset[[var]]))) {
      vars_to_drop <- c(vars_to_drop, var)
      # message(paste("Dropped variable", var, "from fold", i,
      # "for random forest due to mismatched levels"))
      # optional for tracking
    }
  }

  # Remove mismatched variables from trainset_rf and testset_rf
  trainset_rf <- trainset_rf %>% select(-all_of(vars_to_drop))
  testset_rf <- testset_rf %>% select(-all_of(vars_to_drop))

  # Align factor levels in testset_rf to match trainset_rf
  # for remaining factors
  for (var in names(trainset_rf)[sapply(trainset_rf, is.factor)]) {
    testset_rf[[var]] <- factor(testset_rf[[var]],
                                levels = levels(trainset_rf[[var]]))
  }

  # Fit the random forest model with consistent columns
  model <- randomForest(predictors, data = trainset_rf, ntree = 10000)
  testset$Predicted <- pmax(predict(model, newdata = testset_rf), 10)

} else if (method == 'svm') {

```

```

# Fit the SVM model based on specified kernel and type
model <- svm(predictors, data = trainset,
             kernel = svm_kernel,
             type = svm_type)
testset$Predicted <- pmax(predict(model, newdata = testset), 10000)

} else if (method == 'nn') {
  model <- nnet(predictors, data = trainset, size = 5, linout = TRUE,
               trace = FALSE)
  testset$Predicted <- pmax(predict(model, newdata = testset), 10000)
}

```

Intricacies of Methods

```

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

We apply cross-validation with various fold counts (2 to 10) to evaluate the RMSE across different model configurations.

```

# Cross-validation setup
folds_range <- 2:10
methods <- c('linear', 'quadratic', 'rf', 'svm')
results_all_0 <- data.frame(Folds = integer(), Method = character(),

```



```

Mean_RMSE = numeric()

for (folds in folds_range) {
  for (method in methods) {
    cv_result <- cross_validate_rmse(data = train_nomissing,
                                     target = "SalePrice", folds = folds,
                                     methods = c(method))

    cv_result$Folds <- folds
    results_all_0 <- rbind(results_all_0, cv_result)
  }
}

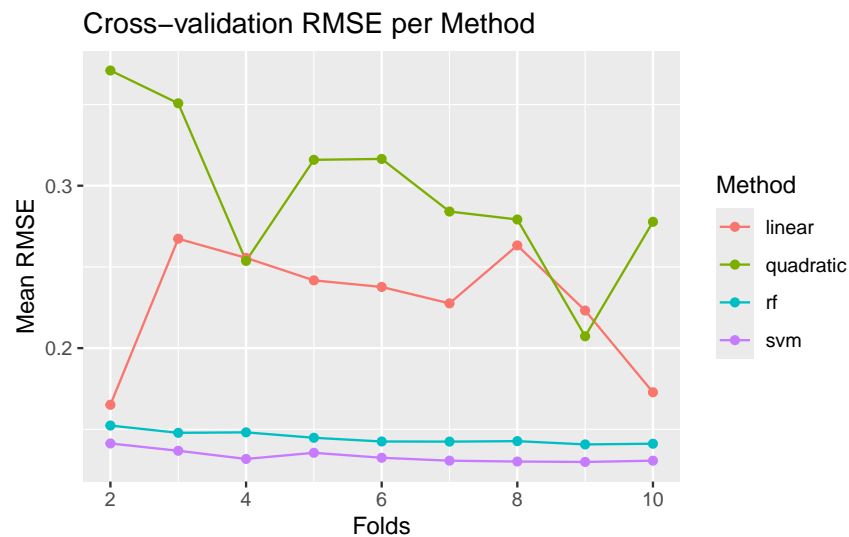
```

Warning in predict.lm(model, newdata = testset): prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

```

# Plot results
ggplot(results_all_0, aes(x = Folds, y = Mean_RMSE, color = Method)) +
  geom_line() + geom_point() +
  labs(title = "Cross-validation RMSE per Method",
       x = "Folds", y = "Mean RMSE")

```



Models Selected

Based on the previous cross-validation, we selected Random Forest and SVM as the best-performing models. Linear and quadratic models are retained for interpretability. **We can**

now compare performance with the model fitted on the whole data with ‘imputed’ values.

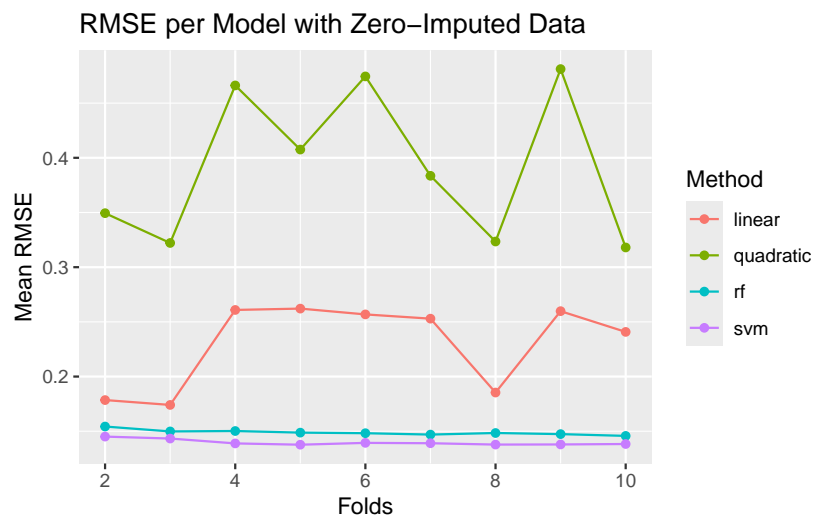
On the Dataset with Zero'd Missing Data

```
# Cross-validation on zero-imputed data
results_all <- data.frame(Folds = integer(), Method = character(),
                          Mean_RMSE = numeric())

for (folds in folds_range) {
  for (method in methods) {
    cv_result <- cross_validate_rmse(data = train_zero, target = "SalePrice",
                                     folds = folds, methods = c(method))

    cv_result$Folds <- folds
    results_all <- rbind(results_all, cv_result)
  }
}

ggplot(results_all, aes(x = Folds, y = Mean_RMSE, color = Method)) +
  geom_line() + geom_point() +
  labs(title = "RMSE per Model with Zero-Imputed Data", x = "Folds",
       y = "Mean RMSE")
```



Folds	Method	Mean_RMSE_no_missing	Mean_RMSE_zero_imputed
-------	--------	----------------------	------------------------

1	10	linear	0.1728179	0.2408181
2	10	quadratic	0.2777971	0.3180323
3	10	rf	0.1411327	0.1457396
4	10	svm	0.1306823	0.1383188
5	2	linear	0.1651218	0.1785062
6	2	quadratic	0.3709527	0.3493832

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.

```
# 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) {

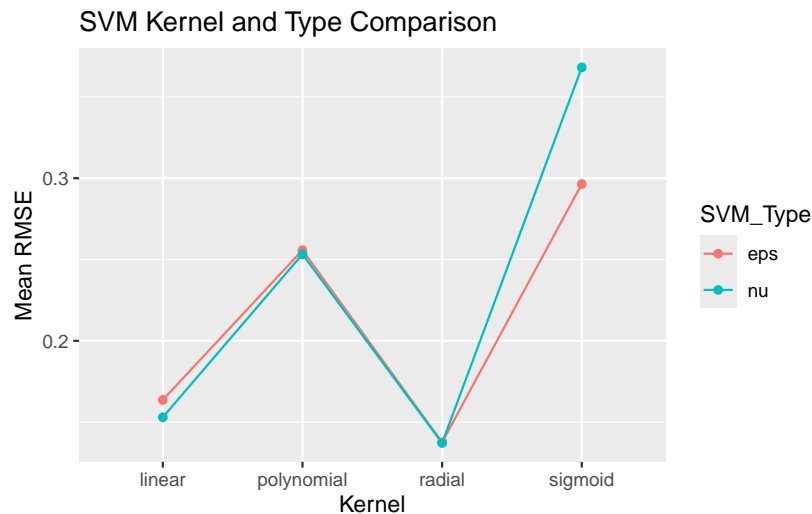
    # Run cross-validation (returns columns Method, Mean_RMSE)
    cv_result <- cross_validate_rmse(
      data = train_zero,
      target = "SalePrice",
      folds = 5,
      methods = c("svm"),
      svm_kernel = kernel,
      svm_type = svm_type
    )

    # 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.


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 = "submission_Adam_Maghout.csv", row.names = FALSE)
```

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

540	Adam Maghout		0.12359	2	15m
-----	--------------	---	---------	---	-----



Your Best Entry!
Your submission scored 0.12414, which is not an improvement of your previous score. Keep trying!

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 and the number of folds.

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](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>