DATA 6100 - Project 1

Chathushi Thalpage (Student ID: 1358582)

1. Overview

In this project, I trained a linear regression model to predict house prices in Ames. The raw housing dataset contained 1000 training houses with 82 columns and 446 test houses with 81 columns. Rather than using more advanced algorithms, I focused on **basic linear regression**, because of its simplicity and interpretability.

2. Data Preprocessing

First of all, I did some initial data exploration to get some idea about the shape of the dataset to identify the numerical data, categorical data, missing values and the distribution of sale price. Then followed below preprocessing steps.

Step 1: Removing Unnecessary Columns

I began by dropping columns such as Id and Unnamed: 0 since they are just index columns and contain no useful information for price prediction.

Step 2: Handling Missing Values

The dataset had several missing values across numeric and categorical columns. I handled them carefully depending on their meaning:

- **Categorical columns** like Alley, BsmtQual, FireplaceQu and PoolQC were filled with 'None', since missing values meant the house didn't have that feature.
- **Numerical columns** like LotFrontag, MasVnrArea were filled with the median of the training data as median is not affected by the extreme outliers.
- **Electrical** had only one missing value and I filled it with the mode ('SBrkr'), since that was the most common value and avoids biasing to the rare categories .

Step 3: Feature Engineering

I added some new features like HouseAge, TotalBath, TotalSF and YearsSinceRemod because raw data doesn't always have predictive power and creating new features makes the model more interpretable.

Step 4: Handling Outliers

To handle the outliers in features, I applied 99th percentile clipping to all numeric features. This means if a feature had an unusual large value, I limited it to a safe upper boundary. This helped the model to predict normal home prices more accurately, making the results more realistic.

Step 5: Merging Rare Categories Under 'Other'

Columns such as Neighborhood, Exterior1st and Exterior2nd had many different categories, and some appeared only once or twice. I grouped these rare categories under "Other" to prevent the model from overfitting to uncommon labels and to reduce the number of unnecessary columns created during one-hot encoding.

Step 6: Ordinal Encoding

I converted quality related columns which represented quality levels (e.g. None, Poor, Fair, Typical, Good, Excellent) into numeric form using a consistent mapping from 0 to 5. This encoding was applied to columns such as ExterQual, BsmtQual, KitchenQual and FireplaceQu.

Step 7: One-Hot Encoding

I used one-hot encoding for other features that don't have natural order (like Neighborhood, GarageType, RoofStyle, etc..). This created a column for each category. Also some of the numerically represented categorical data such as 'MSSubClass, MoSold and YrSold' were converted to String so that they are actually treated as categorical data. This helped the model to understand them as labels instead of numeric values, which prevents wrong mathematical relationships and improves prediction accuracy.

3. Modelling and Model Tuning

Step 1: Train / Validation Split

To evaluate model performance during development, I split the 1000 house training set into:

- Training subset: 800 houses (80%) - used to train models
- Validation subset: 200 houses (20%) - used to test model performance

Step 2: Preselection of features

After encoding, the dataset had 212 columns. Running forward selection on all of them took a very long time, since the method tests each feature one by one in multiple rounds. So before training the model with forward selection, I reduced the number of features by keeping the top 100 variables most correlated with SalePrice (using Pearson correlation). This ensured that forward selection operated only on the most relevant variables.

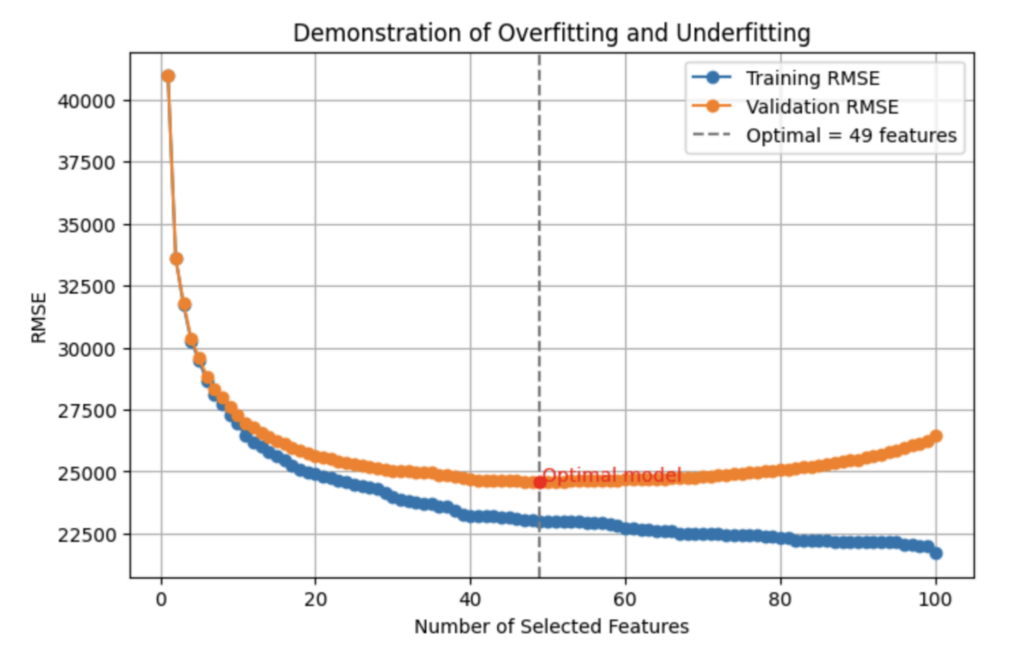
Step 3: Forward Feature Selection

I applied forward selection to find the best combination of variables for the model. This approach added one feature at a time and helped me identify a small group of features that truly matter for predicting house prices while ignoring less important ones. Also I used 5-fold cross validation, as it gives more stable results and reduces the impact of a single random split.

4. Visualizing Overfitting and Underfitting

I plotted training RMSE and validation RMSE versus the number of features added. Initially, both errors decreased, but after a certain point, validation RMSE began to rise, showing overfitting. The optimal number of features where minimum validation RMSE point was found is 49.

Initially, both errors reduced as the model captured important patterns, however when too many variables were added, the model fit the training data better but performed worse on new data showing a clear sign of overfitting. This shows that simpler models can often make more reliable predictions and adding more features does not always improve the performance.



```
Optimal number of features: 49
Training RMSE at best k:  23004.78
Validation RMSE at best k: 24594.21
```

5. Final Model Training & Evaluation

After selecting the best feature set using forward selection and cross validation, I ran the model on the full training data set. The final training RMSE was around $23,850.

Finally, I used the same model to predict prices for the unseen test dataset which was processed with the same features as training. These test predictions represent how the model would perform on real unseen houses. Test RMSE after Submitting to Gradescope was $24,113.

9	Linear only test1	24113