

Ari Schwartz	305656480
Tom Kessous	206018749

# **Statistical Inference and Data Mining**

## **Mid-term Project**

### **1. Introduction**

The purpose of this project is to create a model to predict housing prices using regression techniques learned in class.

### **2. Data Preprocessing**

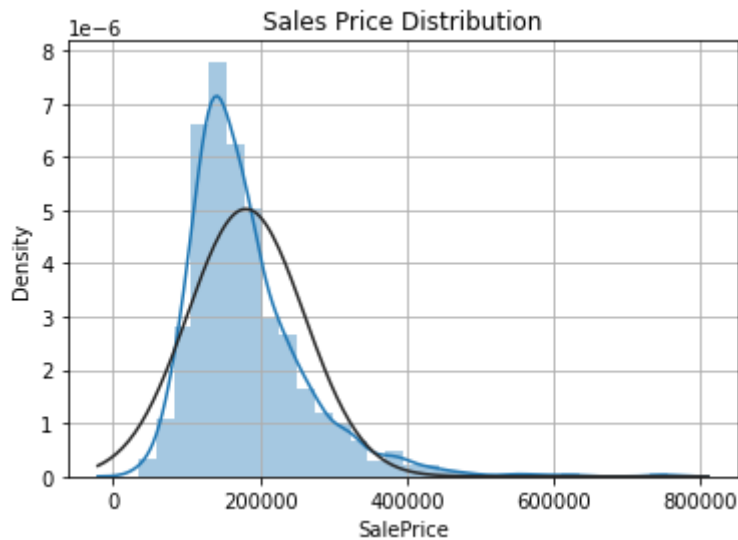
Unfortunately, training a model on real-life raw data isn't a good idea. Real-life data tends to be noisy, have missing values, have irrelevant features and have outliers. For all these reasons (and a few more that we'll see shortly), the data must be processed before it can be used for training.

The first thing we did was explore the data, to get an idea of what we were dealing with.

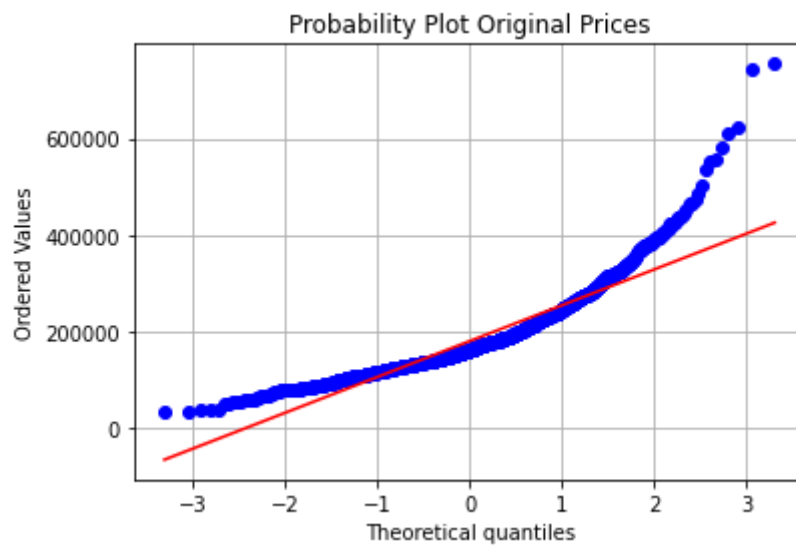
The data consists of 1460 real houses from Ames, Iowa. Each house is described by 79 non-dependant features and the price for which the house was sold (the dependent feature).

## 2.1. Distribution

We began by plotting the house prices to see how they're distributed:

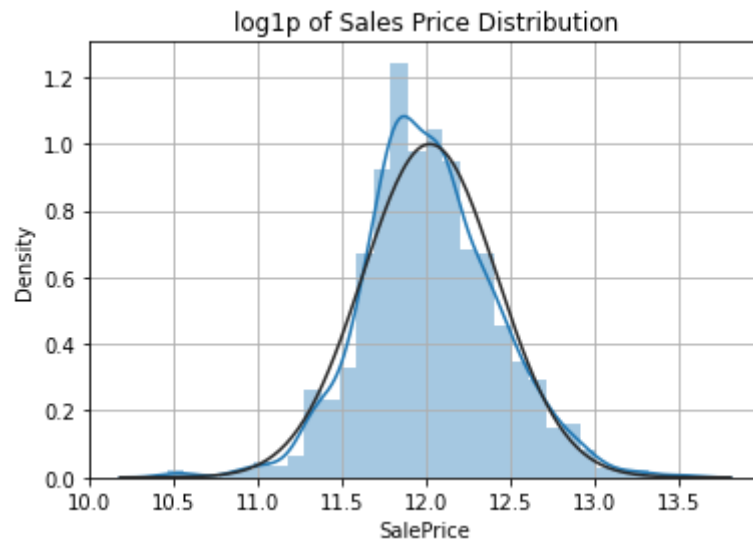


It appears as though the prices are slightly skewed, resulting in a non-normal distribution. This was confirmed using a probability plot (QQ plot):



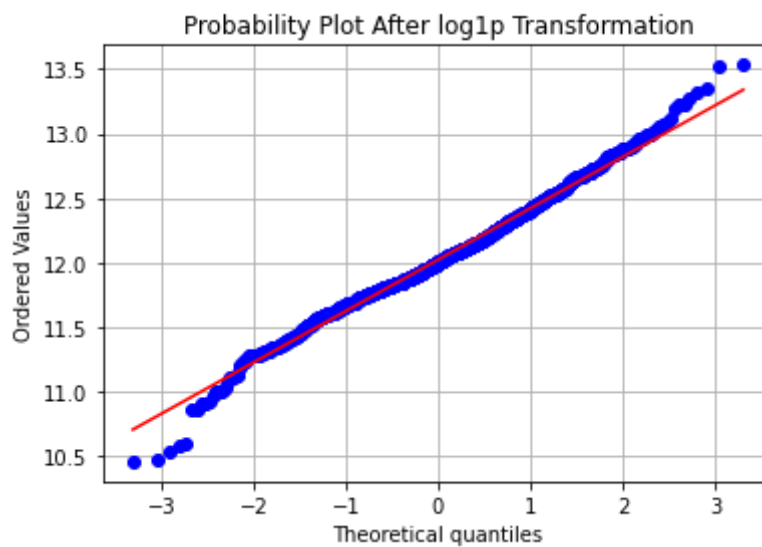
If the prices were normally distributed, we'd expect the probability plot to be linear, but in fact the prices differ significantly from the red line.

To achieve a normal distribution, we decided to apply the log1p transformation:



This plot looks a lot more like a normal distribution.

As expected, the probability plot after the log1p transformation is much closer to linear:



## 2.2. Missing Values

Our next observation was that there are many missing values in the data set. The percentage of missing data points for each feature:

*MSZoning* : 0.14 %  
*LotFrontage* : 16.65 %  
*Alley* : 93.22 %  
*Utilities* : 0.07 %  
*Exterior1st* : 0.03 %  
*Exterior2nd* : 0.03 %  
*MasVnrType* : 0.82 %  
*MasVnrArea* : 0.79 %  
*BsmtQual* : 2.77 %  
*BsmtCond* : 2.81 %  
*BsmtExposure* : 2.81 %  
*BsmtFinType1* : 2.71 %  
*BsmtFinSF1* : 0.03 %  
*BsmtFinType2* : 2.74 %  
*BsmtFinSF2* : 0.03 %  
*BsmtUnfSF* : 0.03 %  
*TotalBsmtSF* : 0.03 %  
*Electrical* : 0.03 %  
*BsmtFullBath* : 0.07 %  
*BsmtHalfBath* : 0.07 %  
*KitchenQual* : 0.03 %  
*Functional* : 0.07 %  
*FireplaceQu* : 48.65 %  
*GarageType* : 5.38 %  
*GarageYrBlt* : 5.45 %  
*GarageFinish* : 5.45 %  
*GarageCars* : 0.03 %  
*GarageArea* : 0.03 %  
*GarageQual* : 5.45 %  
*GarageCond* : 5.45 %  
*PoolQC* : 99.66 %  
*Fence* : 80.44 %  
*MiscFeature* : 96.4 %  
*SaleType* : 0.03 %

There are several different methods for handling missing values. The simplest solution is to remove the entire row wherever there's a missing value (or the entire column if it's missing enough values like PoolQC). However, removing an entire row also removes the non-missing fields, which isn't ideal. It's obviously advantageous to have as much data as possible for training.

Another approach is to replace the missing values. The missing values can either be replaced with 0's (or "none" for strings) or the mean value of the column. The advantage of this solution is that no data is lost in the process.

We chose the second approach. In columns that contain a lot of values, the missing values were replaced with the column's mean value. In columns that don't have sufficient data for calculating the mean value, the missing values were replaced with 0/none.

### **2.3. Feature Manipulation**

There are certain features that can be looked at differently while maintaining the same data. For example, there are 3 "Year" categories. Although they contain the same information, it makes more sense to look at the age of a house rather than the year it was built, for training purposes.

If we look at 2 houses, one built in the year 2000 and one built in the year 2010, that's only a 0.5% difference, but once converted to age, that's a 50% difference. It's also generally preferable to work with smaller numbers when possible.

There are also some features that can be combined to a single feature. For example, there are 4 features that contain information about the size of the porch. The price of the house is more likely to be affected by the porch's overall size and not the enclosed porch area for example. Therefore, all of these features were combined to a single feature.

If these features weren't combined to a single feature, and the model was trained using all 4 of the porch size features, it's possible that the model would be overfit as these features may not be relevant to the house price when looked at separately.

## **2.4. Outliers**

Outliers are data points which differ greatly from the majority of the other data points. In the majority of cases, outliers can be treated as noise in the data, and therefore it's better to not include them in the training data. For this reason, rows with outlier values were removed from the training dataset.

## **2.5. Data Normalization**

The features are represented in many different units (years, ft<sup>2</sup> and dollars for example). A feature that has larger numbers due to the nature of its units shouldn't have more weight than a different feature which is represented using smaller numbers.

Furthermore, it can be more convenient (and more efficient) to work with smaller numbers.

For these 2 reasons, we chose to normalize the data before training, using sklearn's "fit\_transform" function.

## **2.6. Data Encoding**

Obviously, the model requires all data to be numerical values. There are different methods for converting string values into integers. We decided to use the popular "one hot encoding" method (using panda's "get\_dummies" function).

When one hot encoding is applied to a string feature, it replaces the original column with new columns such that one new column is added for each possible string value. Each new column contains binary values indicating whether the original string was this string value or not.

The encoding is obviously applied to both the training and testing data.

### 3. Training

Now that we have a complete, balanced, numeric and normalized dataset, we can move on to training.

It's crucial that we don't use all our data for training, so that we can test the model on never-before-seen data. This is important to avoid overfitting. An overfit model would test very well on the data used for training, but not do so well on the new data.

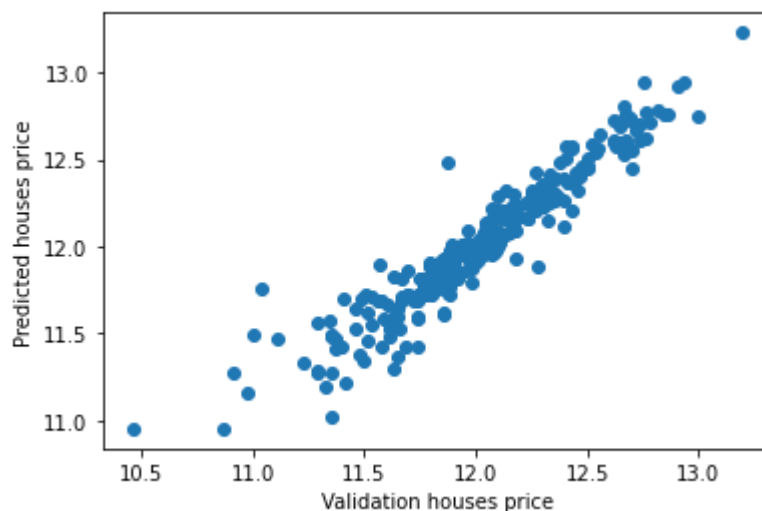
Therefore, we split the training dataset into 2 sets, one for training and one for validation. We shuffle the training set randomly and then put 80% in the training set and 20% in the validation set.

#### 3.1. Models

Next, we need to build a model to predict a house price based on regression methods.

We compared 3 different models, based on different regression techniques:

#### Linear Regression:



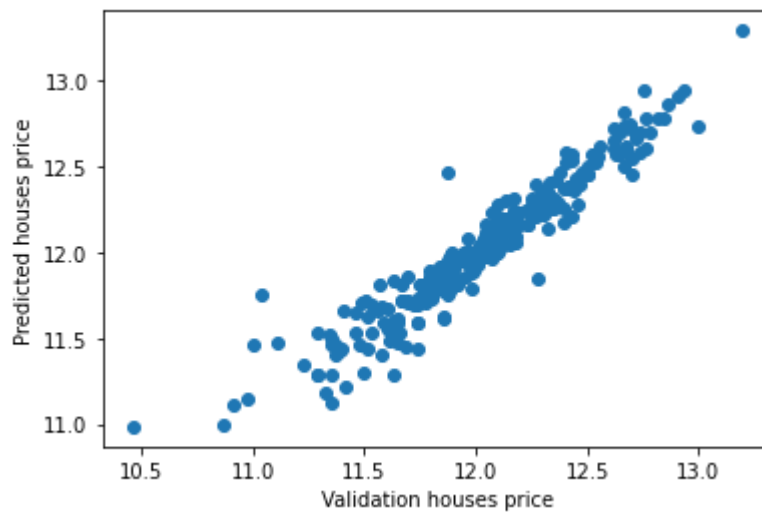
train score( $R^2$ ): 0.9564701172697352

val score( $R^2$ ): 0.904287644815944

train RMSE error: 0.08261561831218378

val RMSE error: 0.12795058860950034

### Ridge Regression:



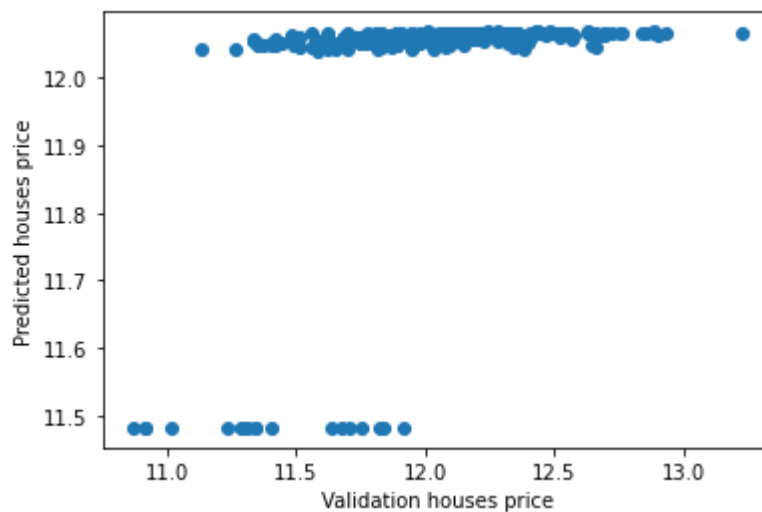
train score( $R^2$ ): 0.9545077597776408

val score( $R^2$ ): 0.9129832933858598

train RMSE error: 0.08445727616440522

val RMSE error: 0.12199993736525554

### Lasso Regression:



train score( $R^2$ ): 0.1089535360670133

val score( $R^2$ ): 0.17389449235161258

train RMSE error: 0.38046587178461505

val RMSE error: 0.3498197593691306



The Lasso model provides poor performance, but that could be because of the hyperparameters. More on that later.

As we can see, Ridge regression provides the best results in terms of R squared ( $R^2$ ) criteria (score) and Root Mean Square Error (RMSE).

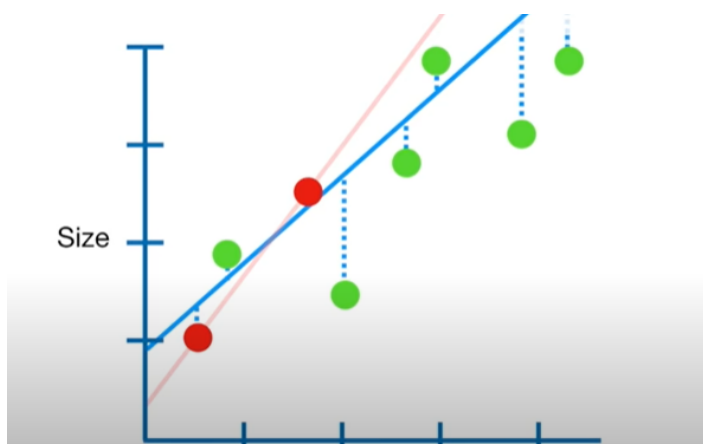
Therefore, we chose Ridge regression as our model.

### 3.2. Ridge Regression

A short theoretical explanation about how Ridge regression works:

Linear regression works by finding coefficients  $W$  which minimize the sum of Least Squares. To be more precise, linear regression actually minimizes the objective function:  $\|y - Xw\|^2$  over L2 norm regularization.

Ridge regression is an extension of linear regression. Ridge regression works by minimizing the objective function:  $\|y - Xw\|^2 + \alpha * \|w\|^2$  over L2 norm regularization. Note that if  $\alpha=0$  we get linear regression. That is, Ridge regression considers the slope squared of the regressor times some  $\alpha$ . That objective function enables the regressor not to pass through all the train points, thus avoiding overfitting the model. In the figure below, the train points are in red and test points in green. Linear regression (the red line) fits the train points, but suffers from overfit due to large error in the test points. Ridge regression (blue line) does not fit the train points (red points) perfectly but estimates the train and test points much better.



In addition, Ridge regression is used for analyzing multiple regression data that suffer from multicollinearity (Multicollinearity, or collinearity, is the existence of

near-linear relationships among the independent variables). When multicollinearity occurs, least squares estimates are unbiased, but their variances are large so they may be far from the true value. By adding a degree of bias to the regression estimates, Ridge regression reduces the standard errors. Hopefully, the net effect will be to estimate more reliably than linear regression.

Ridge regression uses the following formula to estimate coefficients:

$$W = (X'X)^{-1}XY$$

If X is a centered and scaled matrix, the cross product matrix (X'X) is nearly singular when the X-columns are highly correlated, So we can not take the inverse matrix of (X'X). Ridge regression adds a ridge small parameter (k), of the identity matrix to the cross product matrix, forming a new matrix (X'X + kI).

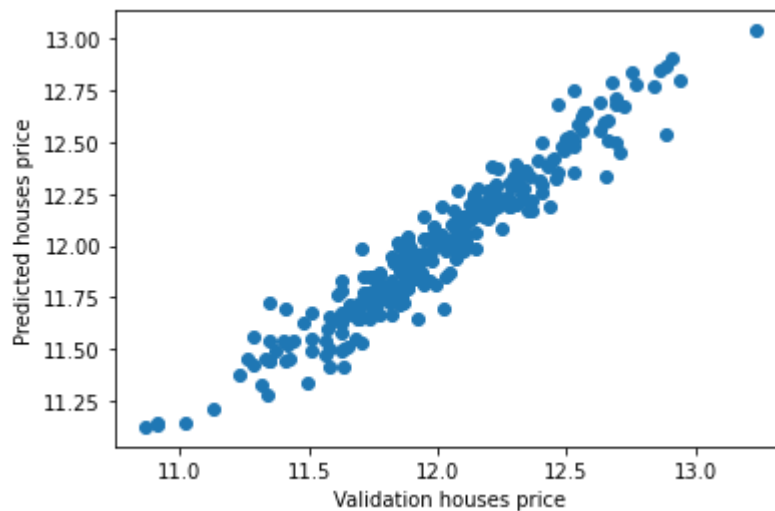
The new formula is used to find the coefficients:

$$W = (X'X + kI)^{-1}X'Y$$

### **3.3. Cross Validation**

In addition, we would like to optimize our model (the Ridge regression) so we decided to perform cross validation (CV) to find the best hyperparameters. We performed cross validation to both the Ridge regression and Lasso regression models, to make sure that Ridge regression still performs better than Lasso regression when both models have optimal hyperparameters.

### Lasso regression CV:



The best alpha is: 0.0006

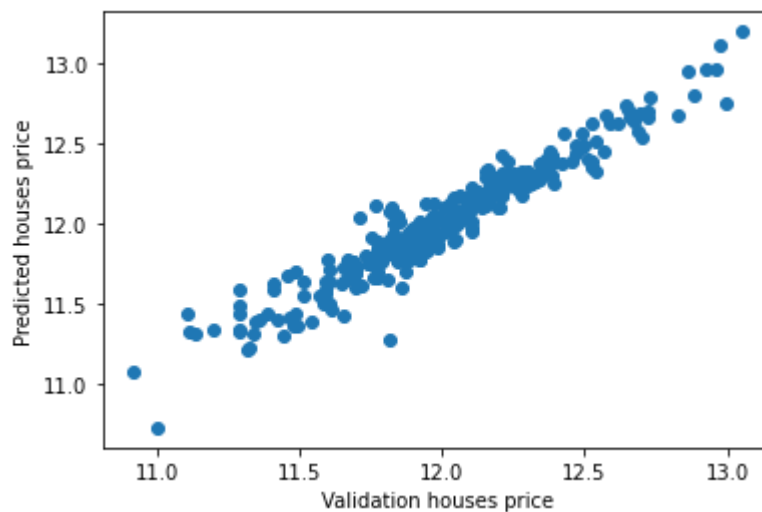
train score( $R^2$ ): 0.9391132822766274

val score( $R^2$ ): 0.9240988861097063

train RMSE error: 0.09945504357405038

val RMSE error: 0.10603534472539844

### Ridge regression CV:



The best alpha is: 11.0

train score( $R^2$ ): 0.9445544624112724

val score( $R^2$ ): 0.9221114599036244

train RMSE error: 0.09549308547936709


val RMSE error: 0.1043940581814867

## 4. Conclusions

As we can see, the Lasso model improved dramatically but the Ridge model was still slightly better.

Cross validation shows that the best alpha is 11. This alpha maximizes R squared ( $R^2$ ) and minimizes the RMSE. Since we shuffled our data randomly each time, the hyperparameters (alpha) might change from time to time.

We submitted our results to the Kaggle site. Our position in the competition is 1707 and our score is 0.13291.

1707	Tom Kessous		0.13291	20	1s
------	-------------	---	---------	----	----