# Supervised Learning: Regression – Peer-graded Assignment – Lego sets

## The Data set

The data source is the Brickset database, which maintains a record of all released Lego sets and related products (<https://brickset.com/sets/category-Normal>).

The data set we are using contains a list of standard released Lego sets, tagged as ‘normal’ in the database, from the years 2011 to 2020. It does not include books, merchandise, magazines and other Lego branded products.

The attributes include the unique identifiers of set ID, set number, variant ID. The theme and sub-theme of the set are included, alongside the set title and year of release.

Counts for the number of minifigures and number of pieces are included, as well as the currency prices in the UK, US, Canada and Euro-zone.

Data on the number of members on the Bricket set website who have flagged the set as one they own, or one they want is also included.

Some categorisation I have personally carried out previously is included to flag if a set is under media license with another company. These flags include Disney, Marvel, Star Wars, Warner Bros, Universal and Other / unknown. This does not included licenses with non-media companies, such as NASA.

In previous work, we have cleaned the raw dataset, carrying out the following:

* Fixing null values with either replacement with the most appropriate values or removal – for example estimating missing set prices in some regions, based on the average conversion rates for sets with known prices
* Removed duplicate data points, such as having several unique identifiers
* Removing unwanted data rows, for example promotional set released, leaving us just with standard set releases
* Converting binary fields to numerical values
* One hot encoding categorical values
* Log transforming skewed data attributes so they become more normally distributed

We will not go into the details of the above data cleaning and preparation here. Instead, the reader can review the additional work at their leisure, if desired. Links are provided to the report and Jupyter Notebook for this work in the Appendix. However, our final cleaned data set has 3982 rows across 268 columns.

## Objectives

It is common for many details about a Lego set, including the number of pieces, theme category and set licenses, to be known about well before the set is officially released and the price is now.

Our primary object in this analysis is try and build a model with our data set that can accurately predict the retail price in the United Kingdom of a Lego set based on all our attributes (excluding the price data in other regions).

We aim to use a range of linear regression techniques to build our model.

While our primary objective will be the accuracy of that model, we will also attempt to interpret our model to identify the most important factors in determining the UK retail price of a Lego set.

## Preparation for the machine learning models

Before we jump in to building our model, we are first going to simplify our data set to enable quicker and simple model building in this stage our work. We will also look at removing several attributes that would have an unwanted impact on the model.

### 3.1 Removal of price data in other regions

### Our objective is to predict the price in the UK. It is reasonably likely that prices will be closely correlated between different regions. It is also highly likely that in the future when trying to use the model to predict the prices of sets before they are released, that prices in no region will be known. We will therefore remove the data on prices in the US, Canada and the EU, leaving on the UK price as our target, ‘y’, variable.

### 

### 3.2 Sub-theme

### It is possible the sub-theme could have impact on our machine learning model. However, given around 180 of 268 columns derive from the categorical sub-theme data, we are removing this data at this stage to initially simplify our model and speed up the data processing. Future work may consider refining the model to contain the sub-theme data.



### 3.3 Number who own the set

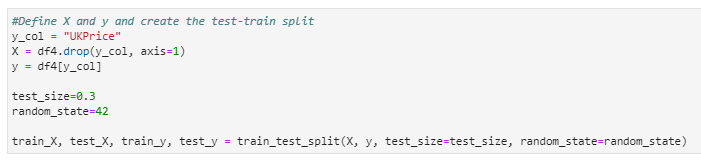
### Before a set is released, no one owns it, therefore building this data point into a model looking to predict a price before released would be wrong and cause a distracting factor in our model when it will often be zero at the point of predicting in future.



## Building the Linear Regression Models

Our dataset now has 75 columns and still has 3982 rows.

Where possible, all our models below used the same test-train splits and same random state. Our test size was set to 0.3, or 30% of our dataset and the random state used throughout was 42.



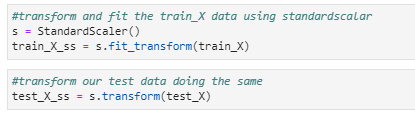
For all the models below, we calculated the R^2 score and mean squared error, we added our coefficients to dataframes, for when we come to interpret the model and we plotted the predicted values against the actual y-values to visually see the results of the model.

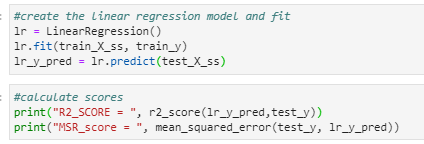
4.1 Linear regression models

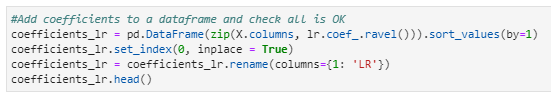
4.1.1 Basic linear regression

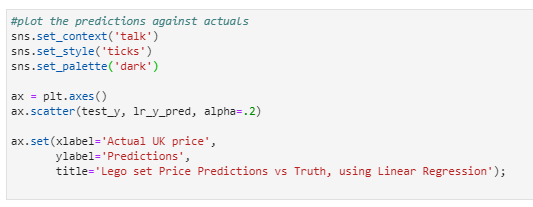
We first standardised our training data using the ‘fit\_transform’ with the ‘standardscalar’ pre-processing operation on the training dataset followed by transforming the test dataset using the same fit.

Then we generated our linear regression model, followed by the steps carried out for all models.







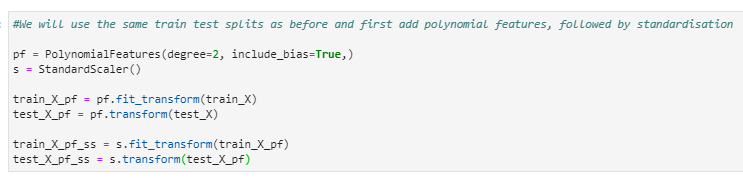


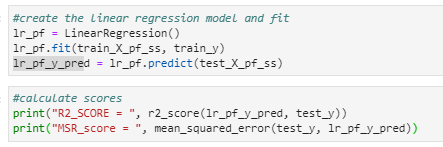
4.1.2 Linear regression with polynomial features

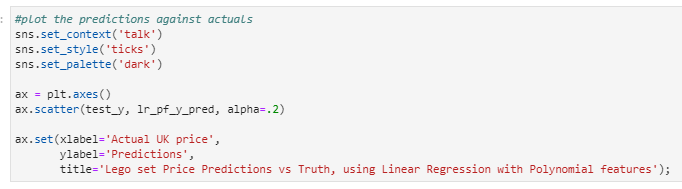
We first added polynomial features of degree 2 to our dataset, followed by applying the standard scalar pre-processing operation, both using ‘fit\_transform’.

Our test set then had the polynomial features and standard scale applied.

Then we generated our linear regression model with polynomial features, followed by the steps carried out for all models.







### 4.2 Cross validation linear regression using k-fold splits

First, we needed to find the most optimal number of folds. We did this by creating a recurrence formula looking at fold numbers from 10 to 50 and using the RepeatedKFold model with 20 repeats.

This generated us a range of R^2 values for each number of folds. We averaged across these to get the average R^2 value for each fold.

The highest average R^2 value was then picked and we used that number to generate our final k-fold model, before carrying out all the steps we did for all models.

The full code for this can be seen in the Python notebook.

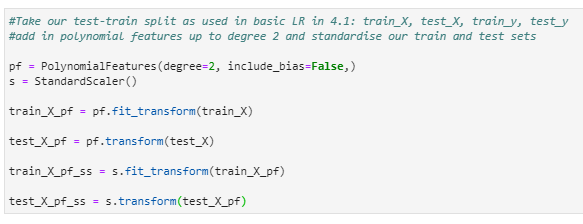
### 4.Lasso with polynomial features

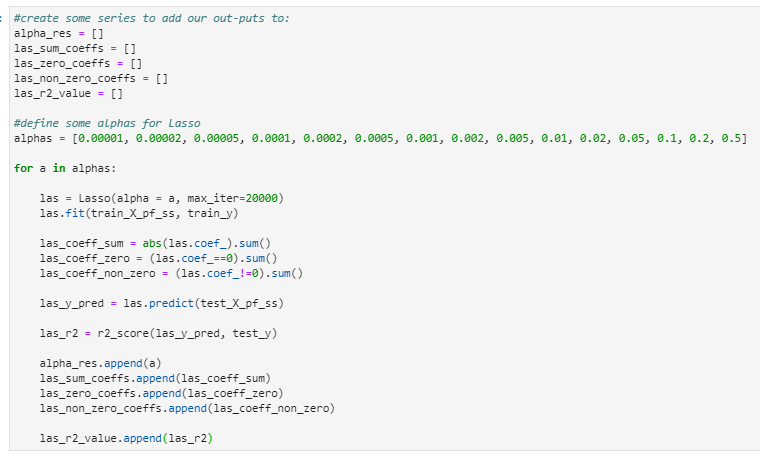
We first added polynomial features of degree 2 to our dataset, followed by applying the standard scalar pre-processing operation, both using ‘fit\_transform’.

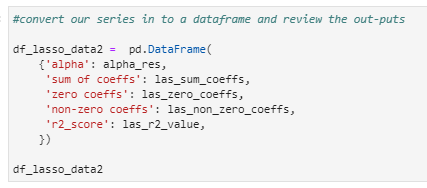
Our test set then had the polynomial features and standard scale applied.

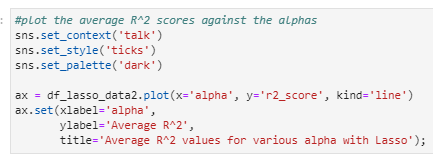
Next, we needed to identify the best alpha value to use.

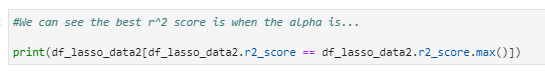
We did this by creating a recurrence formula to apply the Lasso model for 15 different values of alpha ranging from 0.00001 to 0.5 and calculating the R^2 value for each, plotting the outcomes.





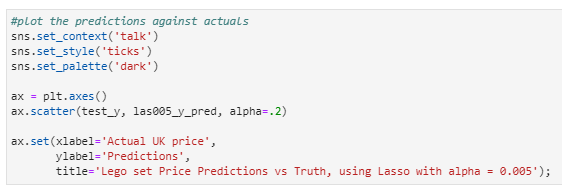






We then took the alpha value which lead to the highest R^2 value and carried out the steps done with all models.





### 4.Ridge Regression with polynomial features

For our final model, we looked at Ridge regression with polynomial features.

We first added polynomial features of degree 2 to our dataset, followed by applying the standard scalar pre-processing operation, both using ‘fit\_transform’.

Our test set then had the polynomial features and standard scale applied.

Next, we needed to identify the best alpha value to use.

We did this by creating a recurrence formula to apply the Lasso model for 12 different values of alpha ranging from 0.0001 to 0.5 and calculating the R^2 value for each, plotting the outcomes.

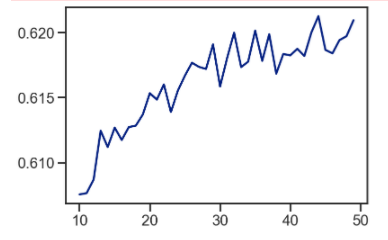
We then took the alpha value which lead to the highest R^2 value and carried out the steps done with all models.

As the code is similar to that included above for Lasso, I have not included it here.

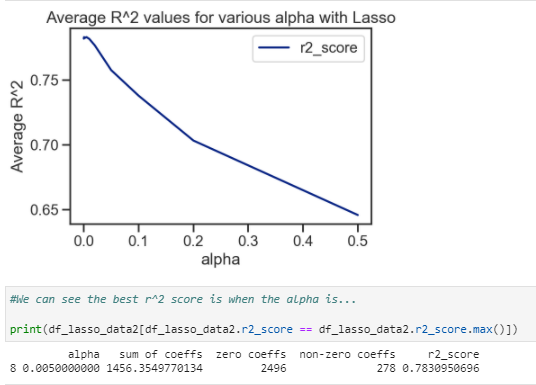
## Findings of the analysis

5.1 Identifying hyperparameters and optimal numbers of folds

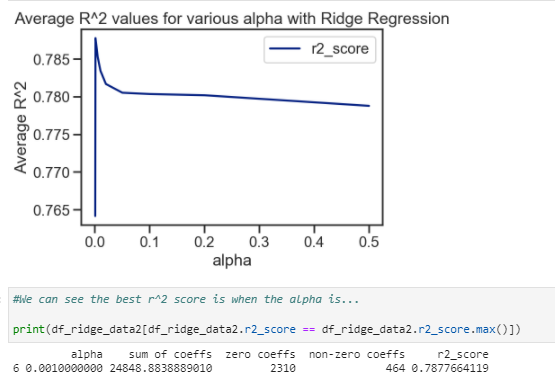
Our analysis highlighted the best value for k in our k-fold linear regression model was 44. With k on the x-axis and R^2 on the y-axis, the outcomes are plotted below:



Our Lasso model with polynomial features identified the optimal alpha value was 0.005:



Our Ridge model with polynomial features identified the optimal alpha value was 0.001:

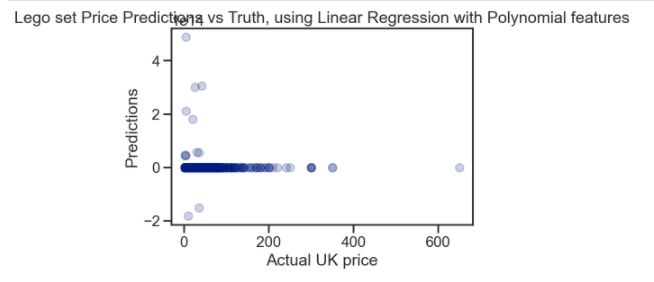


5.2 Identifying the best model

Let us look at a summary of the R^2 values from our models:

|  |  |  |
| --- | --- | --- |
| **Model** | **R^2 value** | **Mean square error** |
| Basic linear regression | 0.39059668 | 761.15382 |
| Linear regression with polynomial features | -0.00290992 | 4.7568x10^26 |
| K-fold linear regression (k=44) | 0.61733682 | 795.08763 |
| Lasso with polynomial features (alpha = 0.005) | 0.78309507 | 355.27721 |
| Ridge with polynomial features (alpha = 0.001) | 0.78776641 | 356.71363 |

We can see there were some major issues with our linear regression model with polynomial features. The negative R^2 should not have been seen and suggests either an issue in our processing, or possible a failed model was created, which did not cross the y-axis. Given the he mean squared error and the fact most Lego sets were predicted a price of £0 by this model (see chart below), it suggests the model failed to be appropriately generated and therefore should be ignored as part of this analysis.

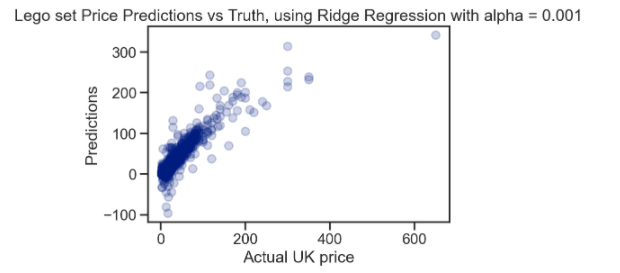


From our remaining models, we can see the basic linear regression model performed worst of all and also significantly worse than the linear regression model with polynomial features (R^2 of 0.3906 compared to 0.6173), suggesting it was correct to include polynomial features in our model.

In addition, we also see that both our regularisation models, which included polynomial features, produced a R^2 score that was higher than our linear regression with polynomial features – Lasso at 0.7831 and Ridge at 0.7878.

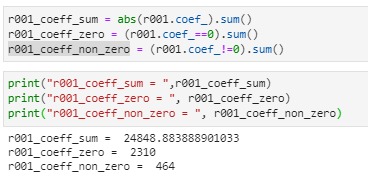
This suggests both of these models, which look at reducing and/or removing coefficients from our model result in our best models in this analysis, with the Ridge model slightly winning out overall.

If we look at the plot of predictions against actual y-values for the Ridge model, below, we can see some interesting predictions where certain lower value Lego sets get negative values. This is clearly impossible and limitation in our model that we should look to remove in further work.



5.3 Interpreting the model

Let us take a look at the coefficients of our Ridge model.



The sum of the coefficients is a little over 24,848. However, we see 2310 are zero, with 464 coefficients having a non-zero value.

However, for interpretability, having 464 non-zero coefficients is not particularly helpful – we will not be able to fully understand what the model is doing.

That said, it is possible to see some basic understandings of the model from the coefficients. We can do this by adding the coefficients to a dataframe allows us to see what features each relate to.



If we look at the largest positive coefficients, we can see that the feature ‘year’ appears in 8 of the top 10 and 14 of the top 20 largest. This makes sense, as it is known that prices rise each year with inflation and also that in recent year a lowering if value in the UK pound against the Euro has seen Lego set price rises in the UK. That the year feature on it’s own has a negative coefficient doesn’t negate this fact – it just suggests that the year feature increases the price only in combination with other factors.

It is interesting to see that ‘pieces’ has the largest negative coefficient at around -956. However, again, we see a string showing for pieces in the polynomial features with positive coefficients. For example – Year x Pieces has a coefficient of just over 725, while Pieces^2 is just over 151. For most Lego sets, this is likely to suggest that more pieces contributions positively to the price making it higher. It’s just the price squared and in combination with other features that has the positive impact, not the price on it’s own.

5.4 Summary of results

Our linear regression models have allowed us to build a Ridge Model with polynomial features with an R^2 score of 0.78776641, giving us a reasonable accurate model for predicting the price of Lego sets in the UK. This is part of our objective that we set out to achieve. However, there is likely room for improvement, as detailed in the final section.

But it is fair to say, our Ridge model has not allowed us to fully understand our model due to the large number of non-zero coefficients in our model. Instead complex interactions between features are at play and only basic interpretation of the model is possible, based on prior intuition and knowledge about our dataset. We have therefore not fully completed this part of our objective in this work.

## Next steps

A major limitation of our models is the prediction of negative prices for some Lego sets. It is clear that we ought to apply, a constraint to our models that negative prices cannot be predicted and re-do the analysis to identify new models.

As we saw the performance of the basic linear regression model was improved by using cross validation of 44 folds, it is possible our attempts at regularisation of a model with Lasso and Ridge regression could also be improved by incorporating cross validation. This could be done using the sklearn RidgeCV and LassoCV linear regression models in Python.

We have also not considered the GridSearchCV ElasticNetCV modelS in this analysis, therefore these would also be another natural extension if looking for alternative and more accurate models, or models which give better interpretability.

Our degree of polynomial extension was only 2 in our work here. It would be interesting to explore the impact higher degrees of polynomial features has on the model. However, this would have to be balanced against processing time required due to the significant increase in the number of features it would introduce. It is likely increasing the degree would further reduce the interpretability.

From our analysis of plotting predicted values against the actual prices, we can see that one Lego set, the Millennium Falcon with price of £649.99 is both significantly higher in price than all other Lego sets in our data set and also sees it’s price substantially under-predicted by our models. It may be considered an outlier and we should consider excluding this data point and re-running our models.

We could also re-introduce some or all of our ‘sub-theme’ features to our data set and re-run some of our models used here to see if these can improve our model. Could this be at the risk of lower interpretability?

Finally, we also know that the theme and sub-threme features both make up the vast majority of the attributes in our data sets. Yet both have categories with very large and very small numbers of rows. Is there a better way of grouping these themes and sub-themes to give us more meaningful data features while also giving us a smaller number of final features to consider? This could be done to allow us to use more complex linear regression models without the current processing time expenses we face or alternatively build models with fewer non-zero coefficients which could increase the interpretability.

## Appendix

1. Report for data cleaning and exploratory data analysis:

<https://github.com/warrenoates1/Coursera_data_science_and_machine_learning/blob/master/Exploratory%20Data%20Anlytics%20for%20Machine%20Learning.docx>

1. Jupyter Notebook for initial data cleaning and exploratory analysis:

<https://github.com/warrenoates1/Coursera_data_science_and_machine_learning/blob/master/Exploratory_Data_Analysis_for_MI_-_Peer-graded_Assignment.ipynb>