case_10.1

May 30, 2020

1 Which salient variables influence the price of homes?

[1]:
```
# Load packages
from IPython.display import Image
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn import linear_model

# This line is needed to display plots inline in Jupyter Notebook
%matplotlib inline
```

1.1 Introduction (5 mts)

Business Context. The modern real estate business is increasingly dominated by large quantities of data. Real estate investment requires sound judgment and accurate price prediction to determine if a property is a suitable investment. Given the illiquid nature of real estate investments, there are a number of complications that can arise when pricing property. Therefore it is of great interest for real estate investors to understand which variables affect home prices so they can plan their investment portfolios with greater clarity and confidence. You recently joined the housing growth division at a multinational corporation, and have been tasked with identifying factors that affect housing prices.

Business Problem. Your task is to answer the following question: **"Which variables are most important for predicting home prices?"**

Analytical Context. The case will proceed as follows: you will 1) explore the data to get some insight into the useful features; 2) apply L1 regularization to reduce the number of variables in your initial linear model; 3) apply L2 regularization to deal with multicollinearity and reduce the variable count further; and finally 4) use both L1 and L2 regularization together via Elastic net regression to get the benefits of both in one model.

1.2 Getting started with the Saratoga home price data (5 mts)

Let's load the data into a `pandas` DataFrame. There are fifteen features that may be used to predict the `price` of a home:

Quantity of Interest: 1. price (millions of US dollars)

Numerical non-categorical features: 1. **lotSize**: size of the lot in acres 2. **age**: age of the house in years 3. **landValue**: value of the land in US dollars 4. **livingArea**: living area in square feet 5. **pctCollege**: percent of the neighborhood that graduated college

Numeric categorical features: 1. **bedrooms**: number of bedrooms 2. **fireplaces**: number of fireplaces 3. **bathrooms**: number of bathrooms 4. **rooms**: number of rooms

Non-numeric categorical features: 1. **heating**: what type of heating the house uses 2. **fuel**: what type of fuel the house uses 3. **sewer**: what type of sewer system the house has 4. **waterfront**: whether or not the house has a waterfront 5. **newConstruction**: whether or not the house has new construction 6. **centralAir**: whether or not the house has a central air system

[2]:
```
# Load and preview data
df = pd.read_csv('saratoga-houses_clean.csv')
df.head()
```

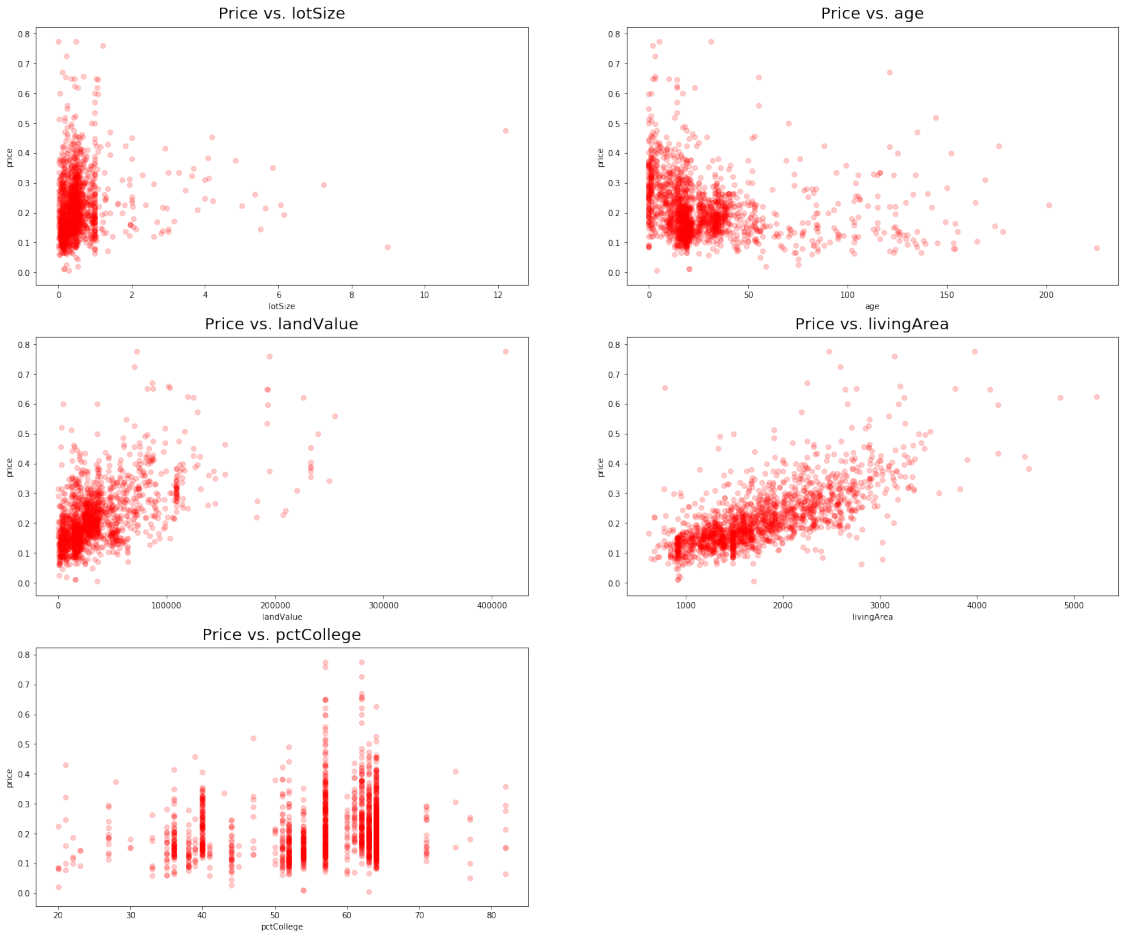
[2]:

	price	lotSize	age	landValue	livingArea	pctCollege	bedrooms	fireplaces	bathrooms	rooms	heating	fuel	sewer	waterfront	newConstruction	centralAir
0	0.132500	0.09	42	50000	906	35	2	1	1.0	5	electric	electric	septic	No	No	No
1	0.181115	0.92	0	22300	1953	51	3	0	2.5	6	hot water/steam	gas	septic	No	No	No
2	0.109000	0.19	133	7300	1944	51	4	1	1.0	8	hot water/steam	gas	public/commercial	No	No	No
3	0.155000	0.41	13	18700	1944	51	3	1	1.5	5	hot air	gas	septic	No	No	No
4	0.086060	0.11	0	15000	840	51	2	0	1.0	3	hot air	gas	public/commercial	No	Yes	Yes

1.3 Exploring feature relationships through data visualization (20 mts)

Let's start by visualizing the relationship between the variables and house prices via 2D scatterplots, beginning with the numeric non-categorical features:

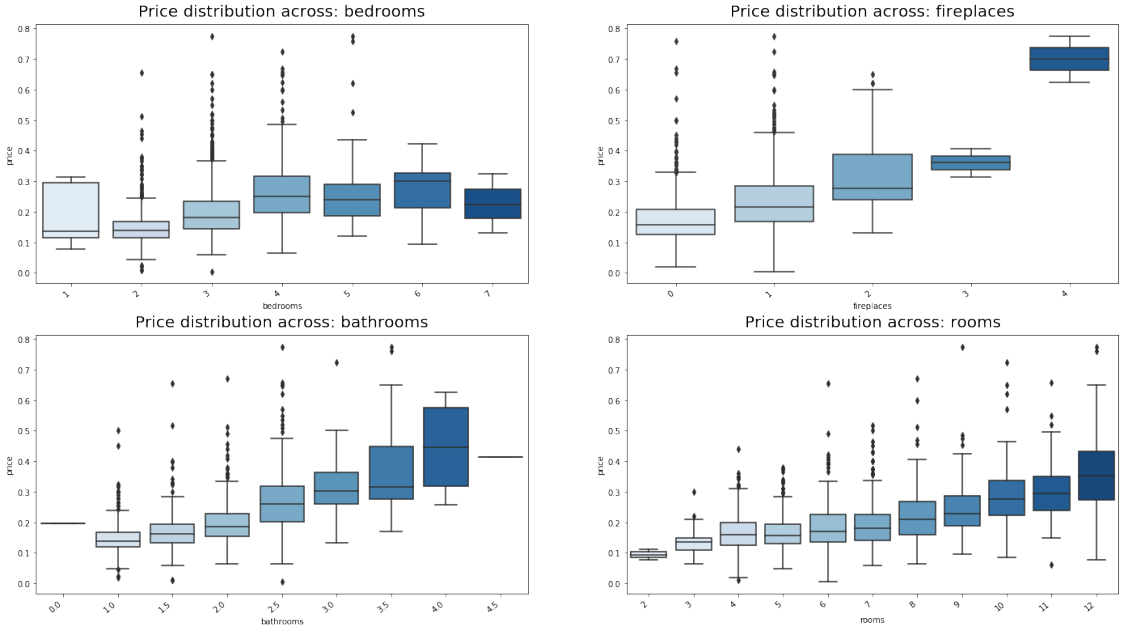
[3]:
```
# Numeric non-catgorical features
plt.figure(figsize=(24,20))
varstolook = ['lotSize', 'age', 'landValue', 'livingArea', 'pctCollege']
for i,feature in enumerate(varstolook):
    plt.subplot(3,2,i+1)
    colvalues = df[feature]
    plt.scatter(colvalues.values, df.price.values, alpha=0.20, edgecolor=None,␣
 ↪color='red')
    plt.xlabel(feature)
    plt.ylabel('price')
    plt.title("Price vs. " + feature, fontsize=20, verticalalignment='bottom');
```



A few observations are evident. First, `livingArea`, `landValue`, and `lotSize` are positively correlated with house prices. Next, the age of a home is inversely correlated with price up to about 50 years, after which the price-age relationship becomes flat. Interestingly, the `pctCollege` variable exhibits clusters around certain percentages; however, there doesn't seem to be any strong relationship between price and `pctCollege`.

Let's look at the numerical categorical features to see if we can notice any significant patterns:

[4]:
```
# Numeric categorical features
plt.figure(figsize=(24,20))
varstolook = ['bedrooms', 'fireplaces', 'bathrooms', 'rooms']
for i,feature in enumerate(varstolook):
    plt.subplot(3,2,i+1)
    pl2 = sns.boxplot(x=feature, y = "price", data = df, palette="Blues")
    pl2.set_xticklabels(pl2.get_xticklabels(), rotation=40, ha="right");
    plt.title("Price distribution across: " + feature, fontsize=20,␣
 ↪verticalalignment='bottom');
```



We immediately notice the strong relationship that `rooms`, `bathrooms`, and `fireplaces` have with home prices. This is expected as larger homes naturally have more rooms and demand a higher asking price. Regarding the `bedrooms` feature, the trend of increasing prices is also present as the number of bedrooms increases, but it is weaker at a low (1) and a high (7) number of bedrooms.

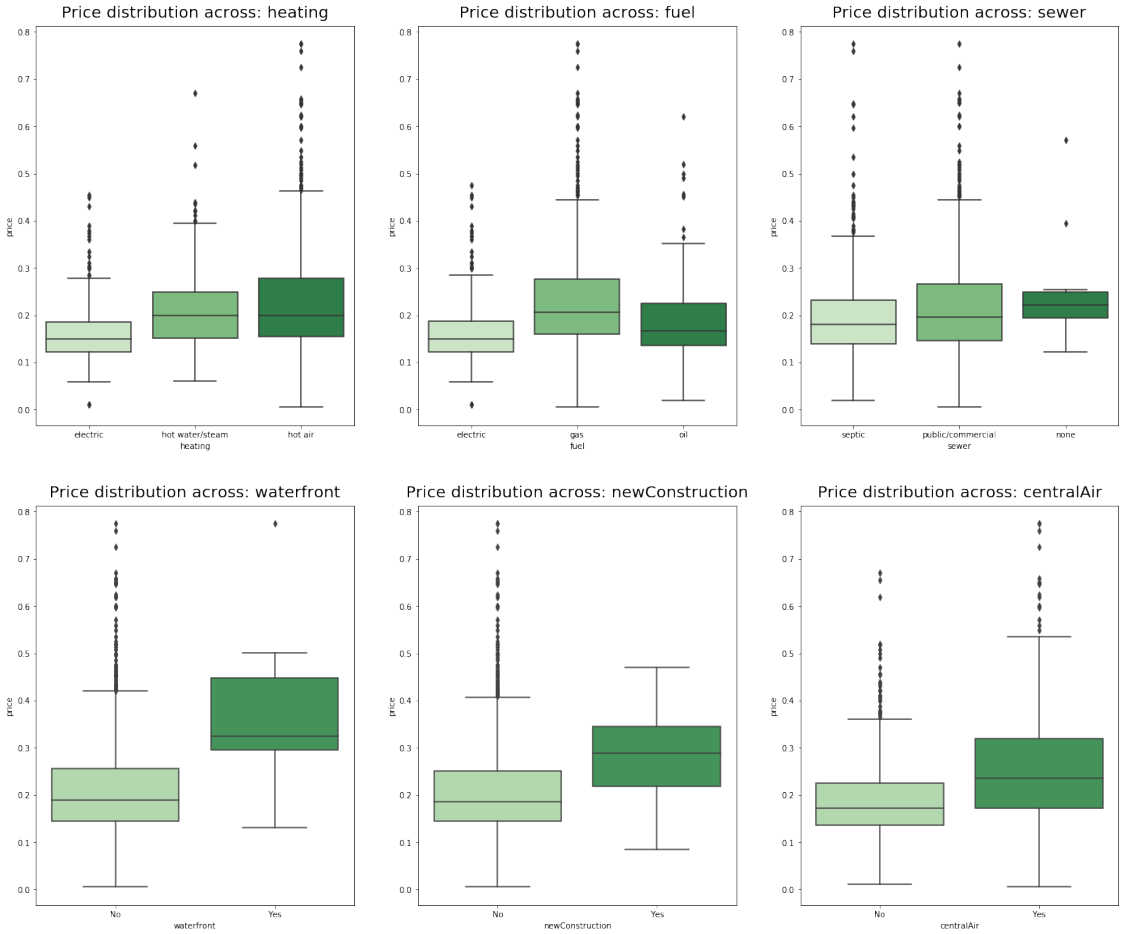
1.3.1 Exercise 1: (10 mts)

Create boxplots to visualize each non-numeric categorical feature against prices for each category value within those categorical features. You should have six plots with boxplots for each category value, one for each non-numeric categorical feature:

1. `heating`
2. `fuel`
3. `sewer`
4. `waterfront`
5. `newConstruction`
6. `centralAir`

Answer. One possible solution is shown below:

[5]:
```
# One possible solution
plt.figure(figsize=(24,20))
varstolook = ['heating', 'fuel','sewer', 'waterfront', 'newConstruction',␣
 ↪'centralAir']
for i,feature in enumerate(varstolook):
    plt.subplot(2,3,i+1)
    pl2 = sns.boxplot(x=feature, y = "price", data = df, palette="Greens")
    pl2.set_xticklabels(pl2.get_xticklabels(), rotation=0, ha="center");
    plt.title("Price distribution across: " + feature, fontsize=20,␣
 ↪verticalalignment='bottom');
```



As expected, a waterfront property, newly constructed units, and homes with central air are all accompanied by higher prices. While weaker, there are some differences across heating, fuel, and sewer variables. Hot air heating, gas fuel, and public/commerical sewer systems charge higher house prices than their adjacent categorical values.

1.3.2 Exercise 2: (5 mts)

There were some relationships that may not have been expected, such as the weak relationship between `pctCollege` and home prices. Which of the following may be a possible external driver of this weak relationship?

A. Neighbourhoods are situated near excellent amenities, driving all the home prices up

B. Neighbourhood residents are diverse and hold an even distribution of income levels and education

C. People tend to live in neighbourhoods where their friends live

D. The data collection procedure for `pctCollege` was taken over the phone

Answer. (B). The explanation is given below.

Answer (A) doesn't immediately lead to any conclusion on why there is a weak relationship between home price and percent college. All (A) implies is that all homes have a higher price due to proximity to attractive amenties and nothing about the distribution of those home prices.

Answer (B) is the ideal choice as having diverse residents in each neighbourhood, in terms of income and education level, will lead to weak relationships in predicting home prices from these quantities. While it may be true that higher income residents own more expensive homes, the percent college variable is relative to a neighbourhood, so an averaging effect will weaken the relationship to price for any given home if the neighbourhood is educationally well-represented.

Answer (C) would likely lead to a stronger relationship between percent college and home prices if it were true in this data set. Friends often have similar interests and may have similar education backgrounds, which could influence neighbourhood percent college averages to become more correlated to home prices.

Regarding answer (D), it tells us nothing about the relationship between percent college and price so it is incorrect. However, it's important to understand the data in this case was acquired by some method. Hence, there may be data errors or bias in the given data. While we will not explore data acquisition methods in this case, if the data was indeed collected over the phone, it would be important to know the response rate by location to determine if each neighbourhood was equally represented.

1.4 Our first model (10 mts)

1.4.1 Preparing the data (5 mts)

Before we begin modeling the data, let's make some simple tranformations. First let's transform the categorical variables with just two categories into binary values where 1 indicates that the effect is present, and zero indicates the effect is not present. This will allow for easy standardization of each feature present in the data:

[6]:
```
# Encoding for non-numeric categorical variables
df2 = pd.get_dummies(df, columns=['heating', 'fuel'], drop_first=False)
df2 = pd.get_dummies(df2, columns=['sewer'], drop_first=True)
df2.replace({'waterfront': {"No":0, "Yes":1},
```

```
                'newConstruction': {"No":0, "Yes":1},
                'centralAir': {"No":0, "Yes":1}}, inplace=True)
```

Next, let's split the data into a training set and a test set. We choose to retain 25 percent of the data as test data:

[7]:
```
# Ready data for multiple regression
X = df2.drop(['price'], axis=1)
y = df2[['price']].values.ravel()

# Split Train and Test Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=0)
```

Finally, let's normalize the data. As we have seen with clustering algorithms, inconsistent scales between variables can negatively affect the performance of various machine learning techniques. It turns out that regularization is one such technique (we will discuss this more later):

[8]:
```
# Get training data mean and standard deviation
training_mean = X_train.mean()
training_std =  X_train.std()

# Center data (common practice when using regularization techniques)
X_train = (X_train - training_mean) / training_std # normalize (use training␣
 ↪mean and training std)
X_test = (X_test - training_mean) / training_std # normalize (use training mean␣
 ↪and training std)
```

1.4.2 Fitting a multiple regression model (5 mts)

Let's start with a simple linear regression model using all of the given features:

[9]:
```
# Fit multiple linear regression to training data
model_linear = sm.OLS(y_train, sm.add_constant(X_train))
original_linear = model_linear.fit()
print(original_linear.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.659
Model:                            OLS   Adj. R-squared:                  0.654
Method:                 Least Squares   F-statistic:                     137.1
Date:                Thu, 28 Nov 2019   Prob (F-statistic):          1.22e-282
Time:                        11:38:48   Log-Likelihood:                1838.2
No. Observations:                1296   AIC:                            -3638.
Df Residuals:                    1277   BIC:                            -3540.
Df Model:                          18
```

```
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.2120      0.002    129.337      0.000       0.209       0.215
lotSize        0.0037      0.002      2.039      0.042       0.000       0.007
age           -0.0052      0.002     -2.593      0.010      -0.009      -0.001
landValue      0.0336      0.002     16.839      0.000       0.030       0.037
livingArea     0.0442      0.003     12.941      0.000       0.038       0.051
pctCollege    -0.0001      0.002     -0.070      0.944      -0.004       0.003
bedrooms      -0.0056      0.002     -2.276      0.023      -0.010      -0.001
fireplaces     0.0007      0.002      0.360      0.719      -0.003       0.005
bathrooms      0.0143      0.003      5.450      0.000       0.009       0.019
rooms          0.0070      0.003      2.643      0.008       0.002       0.012
waterfront     0.0097      0.002      5.776      0.000       0.006       0.013
newConstruction -0.0108    0.002     -5.842      0.000      -0.014      -0.007
centralAir     0.0049      0.002      2.477      0.013       0.001       0.009
heating_electric  0.0006   0.004      0.136      0.892      -0.008       0.009
heating_hot air   0.0018   0.002      0.806      0.421      -0.003       0.006
heating_hot water/steam -0.0028 0.002 -1.350     0.177      -0.007       0.001
fuel_electric  -0.0019      0.004     -0.486      0.627      -0.010       0.006
fuel_gas       0.0019       0.002      0.817      0.414      -0.003       0.006
fuel_oil       -0.0004      0.002     -0.182      0.856      -0.005       0.004
sewer_public/commercial -0.0141 0.011 -1.286     0.199      -0.036       0.007
sewer_septic   -0.0117      0.011     -1.068      0.286      -0.033
```

```
0.010
==============================================================================
Omnibus:                      448.615   Durbin-Watson:                   2.012
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2976.998
Skew:                           1.442   Prob(JB):                         0.00
Kurtosis:                       9.842   Cond. No.                     4.81e+15
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The smallest eigenvalue is 2.67e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

```
/Users/haris.jaliawala/anaconda3/envs/week8/lib/python3.7/site-
packages/numpy/core/fromnumeric.py:2495: FutureWarning: Method .ptp is
deprecated and will be removed in a future version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```

Here, we see that the heat, fuel, and sewer features are not statistically significant, while the living area, land value, waterfront, and new construction variables are signficant. These results are consistent with the strength of relationships that we observed in our previous data visualizations.

1.5 How can regularization help model data? (25 mts)

We have briefly touched upon **regularization** before, and how it is a sort of "penalty" for having too many useless variables in a model. It aims to improve out-of-sample model predictions by reducing model complexity and preventing overfitting. A balanced model will only use enough complexity to capture the general trend of the data; i.e. the minimum number of model parameters possible to get an acceptable fit.

Motivated by the large number of insignificant variables present in the regression, we'd like to reduce our chance of overfitting and remove some of the less useful variables. We will employ a tool called **L1 regularization** to assist us in this goal.

1.5.1 Applying LASSO feature selection (15 mts)

A regression model that utilizes L1 regularization is called a **LASSO (Least Absolute Shrinkage and Selection Operator) regression** model. L1 regularization adds the absolute magnitude of the regression model coefficients (β_j) to the ordinary least squares (OLS) loss function, multiplied by a scaling term λ:

$$\sum_{i=1}^{N} \left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \lambda \sum_{j=1}^{p} |\beta_j|.$$

Recall that mulitple linear regression seeks to minimize the objective function $Loss(Data \mid Model)$, where the loss is usually defined as the squared difference between predicted and true y values (in

this case y is house prices). Extending multiple linear regression to use L1 regularization requires adding an extra term $\lambda \sum_{j=1}^{p} |\beta_j|$ to penalize increased model complexity.

L1 regularization shrinks the β_j coefficients toward zero as any large β_j will greatly increase the above objective function, which we are seeking to minimize. The value of λ sets the strength of the regularization – larger values of λ will increasingly affect the β_j coefficients.

Let's fit a multiple linear regression with an L1 regularization term. In the `linear_model.Lasso()` method the notation `alpha` is referring to the λ value we define above. We first choose a small λ to only perform weak regularization. If any coefficients are set to zero under this weak regularization, then there is a good chance they are not significant to the housing price predictions.

[10]:
```
# Fitting the L1 model
model_l1 = linear_model.Lasso(alpha=0.00001, fit_intercept=True) # higher alpha␣
 ↪for stronger regularization
results_l1 = model_l1.fit(X_train, y_train)
```

Below are the model coefficient names followed by the fit β values. The coefficients are sorted in order from smallest to largest for ease of view. We are looking for the coefficients that are approximately zero as this indicates the L1 regularization has removed these features from the prediction:

[11]:
```
sorted(zip(X_train.columns, results_l1.coef_), key=lambda x: x[1])
```

[11]:
```
[('sewer_public/commercial', -0.013232781041673464),
 ('sewer_septic', -0.010787648875029922),
 ('newConstruction', -0.010751135709249772),
 ('bedrooms', -0.005578301773366284),
 ('age', -0.005197381192771835),
 ('heating_hot water/steam', -0.0032711053509304256),
 ('fuel_electric', -0.001373390224777012),
 ('pctCollege', -0.00011477706533155857),
 ('heating_electric', -0.0),
 ('fuel_oil', -0.0),
 ('fireplaces', 0.0006947901157847524),
 ('heating_hot air', 0.001159923553158023),
 ('fuel_gas', 0.0024158407763989974),
 ('lotSize', 0.0037320106441614837),
 ('centralAir', 0.004876517833592996),
 ('rooms', 0.006969691924555501),
 ('waterfront', 0.009729752501652264),
 ('bathrooms', 0.014290925997177373),
 ('landValue', 0.03358163077689341),
 ('livingArea', 0.044219524339685144)]
```

The results of L1 regularization show the coefficients of **heating_electric** and **fuel_oil** to be zero. This is the idea of L1 regularization applying feature selection. Namely, adding the model complexity constraint to the objective function has forced the fitting procedure to set some of the coefficients to zero (such coefficients we then deem as unimportant).

Let's remove these two variables going forward, and refit a multiple linear regression (no regularization) model:

[12]:
```
# Reduce number of variables based on L1 results
cols_to_drop = ['heating_electric','fuel_oil']
X_train_simplified = X_train.drop(columns=cols_to_drop)
X_test_simplified = X_test.drop(columns=cols_to_drop)
```

[13]:
```
# Re-fit multiple linear regression (no regularization)
model_linear = sm.OLS(y_train, sm.add_constant(X_train_simplified))
res_linear = model_linear.fit()
print(res_linear.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.659
Model:                            OLS   Adj. R-squared:                  0.654
Method:                 Least Squares   F-statistic:                     137.1
Date:                Thu, 28 Nov 2019   Prob (F-statistic):          1.22e-282
Time:                        11:38:48   Log-Likelihood:                 1838.2
No. Observations:                1296   AIC:                            -3638.
Df Residuals:                    1277   BIC:                            -3540.
Df Model:                          18
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.2120      0.002    129.337      0.000       0.209       0.215
lotSize        0.0037      0.002      2.039      0.042       0.000       0.007
age           -0.0052      0.002     -2.593      0.010      -0.009      -0.001
landValue      0.0336      0.002     16.839      0.000       0.030       0.037
livingArea     0.0442      0.003     12.941      0.000       0.038       0.051
pctCollege    -0.0001      0.002     -0.070      0.944      -0.004       0.003
bedrooms      -0.0056      0.002     -2.276      0.023      -0.010      -0.001
fireplaces     0.0007      0.002      0.360      0.719      -0.003       0.005
bathrooms      0.0143      0.003      5.450      0.000       0.009       0.019
```

```
rooms                          0.0070      0.003      2.643      0.008       0.002
0.012
waterfront                     0.0097      0.002      5.776      0.000       0.006
0.013
newConstruction               -0.0108      0.002     -5.842      0.000      -0.014
-0.007
centralAir                     0.0049      0.002      2.477      0.013       0.001
0.009
heating_hot air                0.0011      0.007      0.146      0.884      -0.013
0.015
heating_hot water/steam       -0.0034      0.006     -0.567      0.571      -0.015
0.008
fuel_electric                 -0.0015      0.006     -0.236      0.814      -0.014
0.011
fuel_gas                       0.0024      0.003      0.878      0.380      -0.003
0.008
sewer_public/commercial       -0.0141      0.011     -1.286      0.199      -0.036
0.007
sewer_septic                  -0.0117      0.011     -1.068      0.286      -0.033
0.010
==============================================================================
Omnibus:                      448.615   Durbin-Watson:                   2.012
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2976.998
Skew:                           1.442   Prob(JB):                         0.00
Kurtosis:                       9.842   Cond. No.                         19.6
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

```
/Users/haris.jaliawala/anaconda3/envs/week8/lib/python3.7/site-
packages/numpy/core/fromnumeric.py:2495: FutureWarning: Method .ptp is
deprecated and will be removed in a future version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```

We see the new model has the same R-squared values (to three significant digits). This is important as it indicates that by removing the two features **heating_electric** and **fuel_oil**, we have lost negligible prediction power in terms of explaining variation in the data. Simplifying models like this is very useful in practice and can lead to a better understanding of the effects each feature has on the prediction variable.

1.5.2 Exercise 3: (5 mts)

Reflecting on the use of L1 above, which of the following would be a useful case for L1 regularization?

A. You have data that exhibits large numerical features

B. You have a large number of features and aim to select a smaller number of salient features

C. You have a small number of highly important features

D. You would like to prevent underfitting

Answer. (B). The explanation is given below.

Answer (A) is incorrect as when we prepared the data we removed the mean from each feature. This was specifically done because when using regularization, the magnitude of the β coefficients matter. The added term for model complexity in the model objective function works best when the features are all of similar magnitude so that no feature is given preferential treatment in the fit optimization procedure.

Answer (B) is correct and is how we've used the L1 regularization in this case. We found two features that had their coefficients set to zero upon adding an L1 constraint to the multiple regression model. Thus, we've performed feature selection.

Answer (C) is incorrect as a small number of highly important features does not indicate the need for L1 regularization. L1 regularization is regularly applied for feature selection to shrink some of the coefficients to zero. If all features are known to be very important, L1 regularization may not be the best choice.

Answer (D) confuses underfitting with overfitting. Regularization techniques are applied to prevent overfitting by reducing model complexity.

1.5.3 Identifying multicollinearity and removing more features (5 mts)

Next, let's take a look and see if multicollinearity is present in the features. Recall that multicollinearity is often a source of unnecessary complexity in a model which can lead to overfitting, so it makes sense to screen for it. Let's take a look at correlations among the numerical non-categorical features via a correlation matrix:

[14]:
```
# Create correlation heatmap for continuous variables
features = ['lotSize', 'age', 'landValue', 'livingArea', 'pctCollege']
fig, ax = plt.subplots(figsize=(8,6))
corr = X_train_simplified[features].corr()
sns.heatmap(corr, xticklabels=corr.columns,yticklabels=corr.columns,␣
↪cmap="Blues", annot=True, fmt=".2f");
```

Most correlations not along the main diagonal are low, but we see that there is a sizable correlation between land value and living area. While not a terribly high correlation at 0.45, this is not ideal from a linear regression standpoint and indicates multicollinearity is present.

L1 regularization often suffers with predictions when the data exhibits mutlicollinearity as the optimization algorithm doesn't work as well. Luckily, there is another type of regularization, **L2 regularization**, which is very good at addressing multicollinearity. Let's proceed to apply L2 regularization to the current set of features.

1.6 Applying L2 regularization to address multicollinearity (25 mts)

A linear regression model that utilizes L2 regularization is called **Ridge regression**. L2 regularization adds the squared magnitude of the regression model coefficients (β_j) to the OLS loss function:

$$\sum_{i=1}^{N} \left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2.$$

L2 regularization also shrinks the β_j coefficients toward zero, however the effects of shrinkage are typically less than LASSO regression. Let's fit the model and examine the new coefficients:

[15]:
```
# Fitting the model
model_l2 = linear_model.Ridge(alpha=28.0, fit_intercept=True) # higher alpha␣
 ↪for stronger regularization
results_l2 = model_l2.fit(X_train_simplified, y_train)
```

Below are the model coefficient names followed by the fit β values. The coefficients are sorted from smallest to largest:

[16]:
```
sorted(zip(X_train_simplified.columns, results_l2.coef_), key=lambda x: x[1])
```

[16]:
```
[('newConstruction', -0.010097504317072278),
 ('sewer_public/commercial', -0.00564552678257625),
 ('age', -0.005192786144608434),
 ('bedrooms', -0.004616405925118789),
 ('sewer_septic', -0.003313385633993632),
 ('heating_hot water/steam', -0.00283756265793467),
 ('fuel_electric', -0.0011615800651067435),
 ('pctCollege', 6.581924953499052e-05),
 ('fireplaces', 0.0012042793203153844),
 ('heating_hot air', 0.0016736767194718868),
 ('fuel_gas', 0.002269941382553941),
 ('lotSize', 0.003852235288187061),
 ('centralAir', 0.005066594564126534),
 ('rooms', 0.007718915340688895),
 ('waterfront', 0.009718016717086515),
 ('bathrooms', 0.014790009073400263),
 ('landValue', 0.03313699635793972),
 ('livingArea', 0.04142858301459809)]
```

Here, we see `livingArea` and `landValue` have the largest coefficients. This was expected given that the original data visualization exploration of this analysis showed a significant relationship between these variables and housing prices.

However, it is important to ask why we chose to use $\lambda = 28.0$ in this fit. The value of λ may significantly affect the model coefficients so it must be chosen with care. The following section explores one method for graphically choosing an optimal λ value.

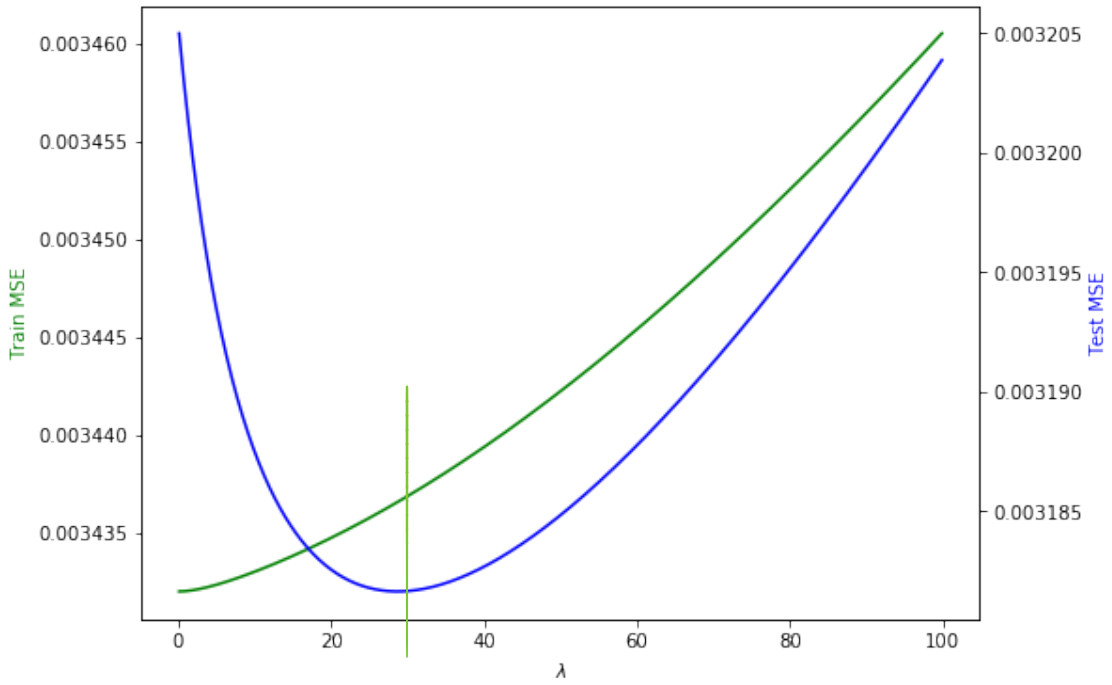
1.6.1 Looking at the effects of λ on underfitting and overfitting (20 mts)

The mean-squared-error (MSE) of the model is calculated by comparing the model predictions to the true values. Here, this means comparing the predicted house prices from the L2 regularized model to the actual house prices. Lower values of mean-squared-error indicate better model performance.

Let's sweep through λ values and see the effects on the model performance. We will compare performance on training data and test data. Recall that higher λ values lead to stronger regularization:

[17]:
```
# Loop through different lambda values
lambdas = np.arange(0.1, 100.0, 0.1)

coefs = []
mse_train = []
mse_test = []
for l in lambdas:
    ridge = linear_model.Ridge(alpha=l, fit_intercept=True)
    ridge.fit(X_train_simplified, y_train)
    mse_train.append(mean_squared_error(y_train, ridge.
 ↪predict(X_train_simplified))) # train data
    mse_test.append(mean_squared_error(y_test, ridge.
 ↪predict(X_test_simplified))) # test data
```

[18]:
```
# Plot results
fig, ax1 = plt.subplots(figsize=(8,6))

ax2 = ax1.twinx()
ax1.plot(lambdas, mse_train, 'g-')
ax2.plot(lambdas, mse_test, 'b-')

ax1.set_xlabel('$\lambda$')
ax1.set_ylabel('Train MSE', color='g')
ax2.set_ylabel('Test MSE', color='b')

plt.show()
```



overfitting

underfitting

Here, we see the model is slightly overfitting when λ is small. The test MSE is much higher than the train MSE and the model is mostly unregularized (i.e. the model is linear regression with no extra terms in the optimization function).

The model is underfitting when λ is large. The effect of a large λ is to increase the regularization strength, effectively make the model restricted in what coefficients it can use to fit the data. With a large λ we see the model struggles on both the train and test data (high MSE for both train and test data), indicating that the model is underfitting.

Thus, we need to choose a lambda that balances overfitting and underfitting. From the test MSE curve, we see the minimum is achieved around $\lambda = 28.0$. This was the motivation in choosing the λ in the previous section. In general, we recommend using such a graphical method to gain an understanding of the effects regularization has on a multiple regression model.

1.6.2 Exercise 4: (5 mts)

In L2 regularization, as λ gets closer to zero, how would the house price model predictions behave? Compare to the case where we fit a multiple linear regression model with no regularization.

A. The house price predictions would become increasingly biased

B. The house price predictions would become less resistant to multicollinearity present in the features

C. The house price predictions do not change as λ changes

D. The model would become less complex, leading to lower variance in predictions

↓ regularización ↓ bias

Answer. (B). The explanation is given below.

Explanation: Answer (A) is incorrect since as λ becomes closer to zero, this is effectively reducing the strength of the regularization (thus removing model bias).

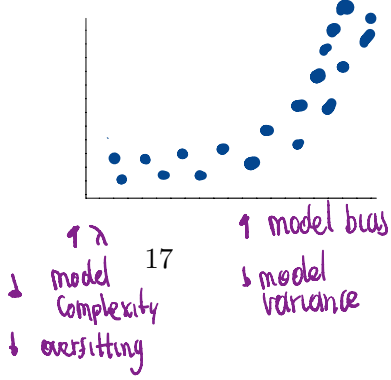
Answer (B) is correct as reducing the λ reduces the strength of the regularization. L2 regularization is primarily used to reduce the effects of multicollinearity by controlling the model complexity. Higher values of λ lead to a more regularized model, thus lesser values of λ will become less resistant to multicollinearity.

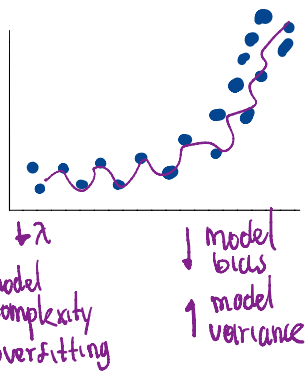
Answer (C) is incorrect since the predictions will change as λ changes since the regularization is part of the model's objective function. The objective function is minimized during model fitting, and because λ is part of the optimization procedure, changing its value will also change the resulting model, and thus change the model estimates.

Regarding answer (D), decreasing λ leads to a more complex model. Through increasing λ the model becomes less complex, thus helping aid against overfitting.

↑ λ ↓ model complexity → ↓ overfitting

↓ λ ↑ model complexity





↑ model variance → cause to model the random noise

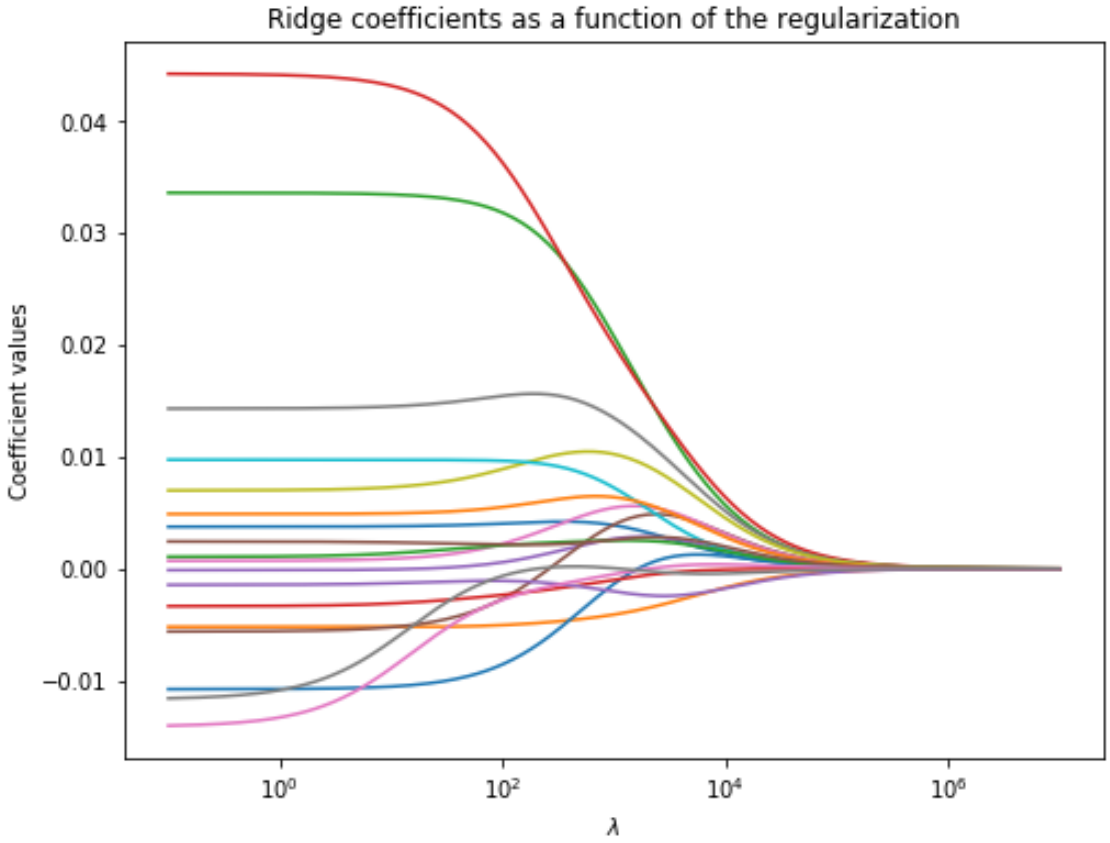
↑ bias → the algorithm miss the relevant relations.

1.6.3 Exercise 5: (10 mts)

We've fit a L2 regularized multiple linear regression to predict housing data and looked at how the chosen λ affects the train and test error. We'd like to see how the coefficients are affected by the choice of λ as well. Write a script to change the value of λ and look at how the fit Ridge regression coefficients change. Namely, write a script to produce the following plot. Each line of the plot corresponds to a single feature's coefficient as the value of λ changes.

[19]:
```
Image("ridge_coef_plot.png", width=500)
```

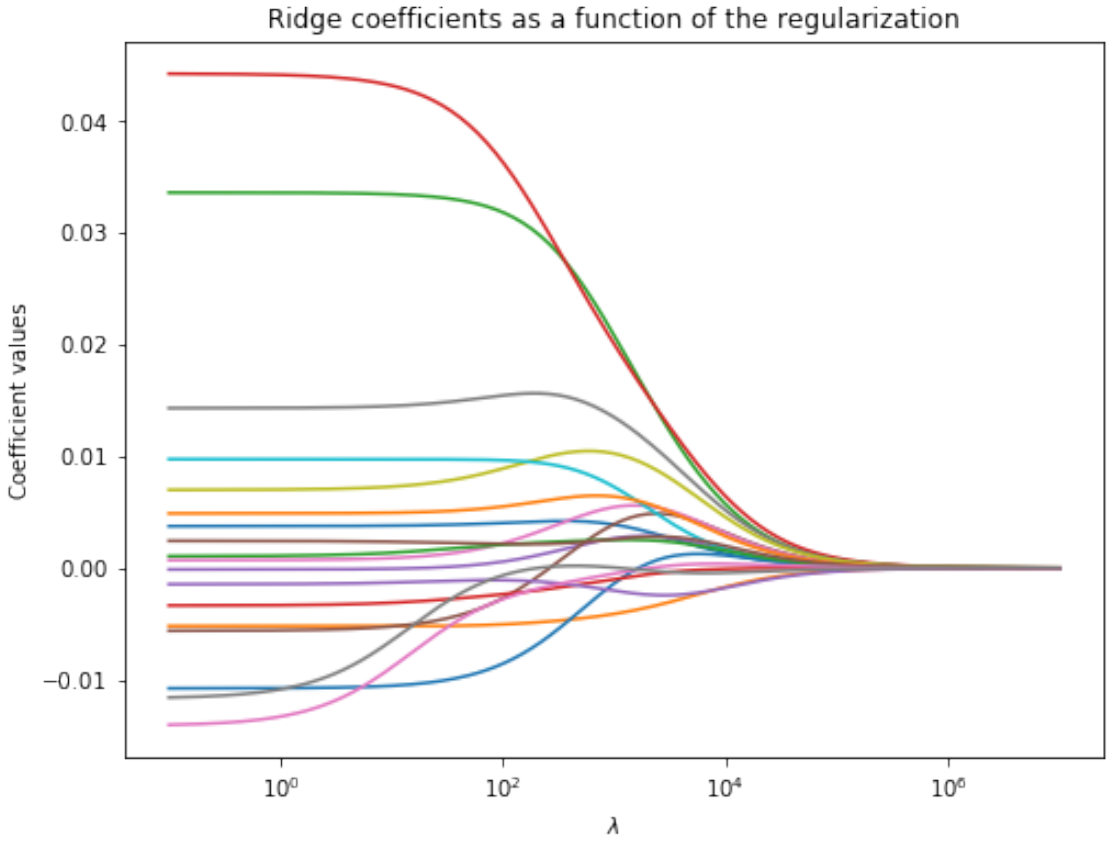
[19]:



Answer. One possible solution is shown below:

[20]:
```
# Loop through multiple values of lambda and record fit coefficients
lambdas = np.logspace(-1, 7, 200)

coefs = []
for l in lambdas:
    ridge = linear_model.Ridge(alpha=l, fit_intercept=True)
    ridge.fit(X_train_simplified, y_train)
    coefs.append(ridge.coef_)
```

```
fig,ax = plt.subplots(figsize=(8,6))
ax.plot(lambdas, coefs)
ax.set_xscale('log')
ax.set_xlim(ax.get_xlim())  # reverse axis
plt.xlabel('$\lambda$')
plt.ylabel("Coefficient values")
plt.title('Ridge coefficients as a function of the regularization')
plt.axis('tight')
plt.show()
#plt.savefig('ridge_coef_plot.png', bbox_inches='tight')
```



Each line above corresponds to one coefficient's path as λ increases from left to right. We see that as the regularization strength increases, all the coefficients eventually become zero. This is the shrinking effect of regularization, where a very large λ will force all model coefficients to zero (due to the optimization prodedure minimizing the objective function). Having all the coefficients go to zero will lead to underfitting as the model predictions become too restricted.

In terms of decreasing λ to very small values, it's clear the model coefficients eventually reach a constant value for small values of λ. This illustrates that as λ becomes small, the regularization essentially disappears and the coefficients

will be the same as if no regularization was applied at all. Depending on the data, this could lead to overfitting.

Thus, there is a trade-off between overfitting and underfitting when choosing an optimal λ value. Too high a λ may underfit, while too low a λ may overfit. By choosing the appropriate λ through careful observation and analysis, regularization techniques can be a powerful tool to prevent and control overfitting.

1.7 Elastic Net regularization (35 mts)

Now that we've applied L1 and L2 regularization individually, is there a method by which we can gain their benefits together? Fortunately, the sparsity benefits of L1 and robustness and prevention against overfitting benefits of L2 can be shared in a method named **Elastic Net regression**.

L1 regularization adds the absolute values of the betas to the objective function:

$$Loss(Data \mid Model) + \lambda \sum_{j=1}^{p} |\beta_j|.$$

On the other hand, L2 regularization adds the squared magnitudes of the betas to the objective function:

$$Loss(Data \mid Model) + \lambda \sum_{j=1}^{p} \beta_j^2.$$

Elastic net can be a powerful method to encourage both a sparse solution and a solution that is robust against detrimental characteristics like multicollinearity. Given the data $(y_1, ..., y_n)$ and the observed features for each y_i, namely $(x_{i1}, ..., x_{ip})$, a regression model with p parameters $(\beta_0, ..., \beta_p)$ takes the form,

$$y_i = \sum_{j=1}^{p} x_{ij}\beta_j.$$

Elastic net regularization blends both L1 and L2 regularization penalties to include both the absolute value and squared magnitude of the beta coefficients in the loss function:

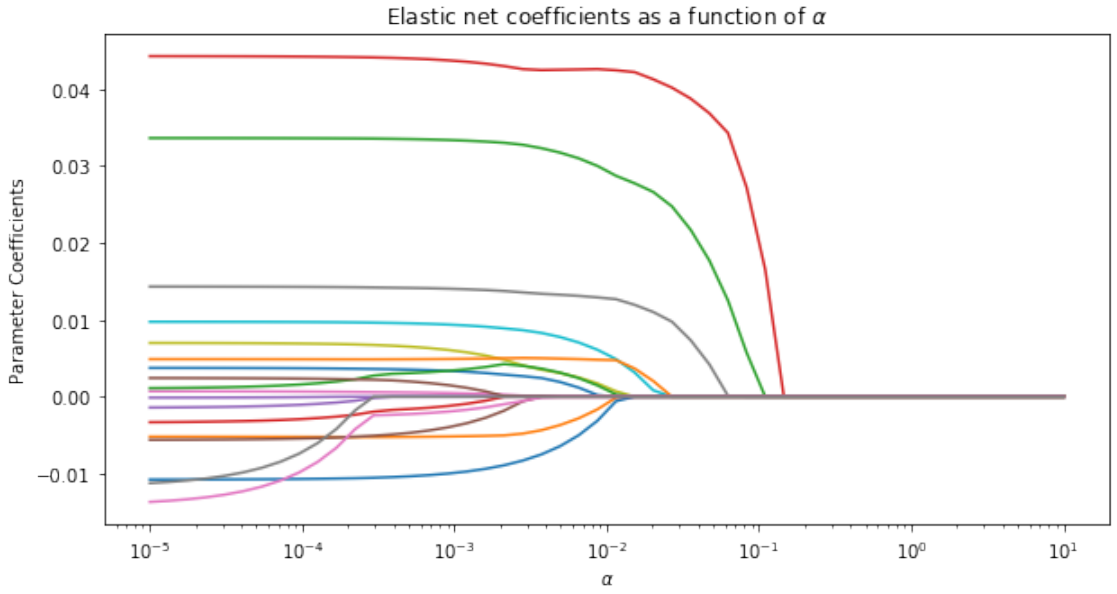
$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \alpha\rho \sum_{j=1}^{p} |\beta_j| + \alpha(1-\rho) \sum_{j=1}^{p} \beta_j^2.$$

Here, α is the regularization rate, and ρ controls the balance between L1 and L2 regularization in the loss function. Namely, $\rho = 1$ results in a pure L1 penalty, $\rho = 0$ results in a pure L2 penalty, while $0 < \rho < 1$ results in a blended elastic net regularization.

Similar to how we calibrated λ in our discussion of L2 regularization, it is helpful to construct graphs as a function of the elastic net regularization rate α. Let's sweep through a list of different α values and plot the coefficients as a function of α:

[21]:
```
# Loop through different regularization alpha values and record fit␣
↪coefficients at each value
alpha_list = np.logspace(-5, 1, 50)

coefs = []
for a in alpha_list:
    enet = linear_model.ElasticNet(random_state=0, alpha=a, l1_ratio=0.5,␣
↪fit_intercept=True)
    enet.fit(X_train_simplified, y_train)
    coefs.append(enet.coef_.ravel())
```

[22]:
```
# Plot results
fig, ax = plt.subplots(figsize=(10,5))
ax.plot(alpha_list, coefs)
ax.set_xscale('log')
ax.set_xlim(ax.get_xlim())  # reverse axis
plt.xlabel(r'$\alpha$')
plt.ylabel('Parameter Coefficients')
plt.title(r'Elastic net coefficients as a function of $\alpha$')
plt.axis('tight');
```



From the above plot, we see that as α increases beyond ~0.1, all the coefficients become zero. This indicates the regularization at such an alpha is too strong. On the other hand, very small α values lead to a weak regularization. We can see as the α values decrease from right to left, the parameter estimates start to be more significantly affected near 10^{-4}. This incidates at that point the regularization starts to have an effect on the model fit.

Now that we have an idea of how the coefficients are affected by α, let's take a look at how to choose the optimal α value.

1.7.1 A method to choose the optimal α (10 mts)

Choosing an optimal α is important. Choosing an α that is too large will cause the coefficients of the model to become too constrained and reduce out-of-sample model accuracy. On the other hand, choosing an α that is too small can result in the regularization having little effect, thus reducing the primary purpose of regularization as a control against overfitting.

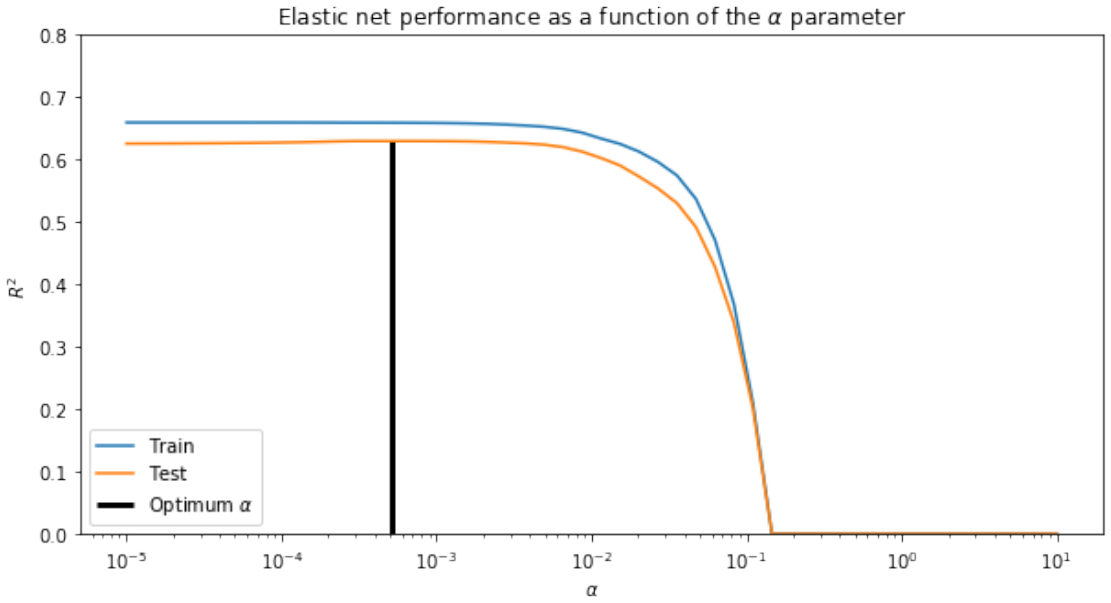
One method to select an optimal α is to observe the R-squared of the model as a function of α, and subsequently select the α that gives the highest test set R-squared value. Let's fit the elastic net regularized linear regression model for different values of α and see the effects on the R-squared for both training and test sets. Note that we will set the ratio of L1 and L2 regularization (denoted as the l1_ratio parameter in linear_model.ElasticNet(), and often denoted mathematically as ρ) to 0.5 in this analysis, with the idea of balancing L1 and L2 regularization effects:

[23]:
```
# Compare progression of train and test errors and alpha varies
alphas = np.logspace(-5, 1, 50)
enet = linear_model.ElasticNet(random_state=0, l1_ratio=0.5, fit_intercept=True)
train_errors = list()
test_errors = list()
for alpha in alphas:
    enet.set_params(alpha=alpha)
    enet.fit(X_train_simplified, y_train)
    train_errors.append(enet.score(X_train_simplified, y_train))
    test_errors.append(enet.score(X_test_simplified, y_test))

i_alpha_optim = np.argmax(test_errors)
alpha_optim = alphas[i_alpha_optim]
print("Optimal alpha regularization parameter : %.6f" % alpha_optim)

# Estimate the coef_ on full data with optimal regularization parameter
enet.set_params(alpha=alpha_optim)
coef_ = enet.fit(X, y).coef_
```

Optimal alpha regularization parameter : 0.000518

Let's see how the R-squared of the training and test data evolves as the α parameter is changed:

[24]:
```
# Plot R-squared as a function of alpha
fig,ax = plt.subplots(figsize=(10,5))
plt.semilogx(alphas, train_errors, label='Train')
plt.semilogx(alphas, test_errors, label='Test')
plt.vlines(alpha_optim, plt.ylim()[0], np.max(test_errors),␣
 ↪color='k',linewidth=3, label=r'Optimum $\alpha$')
plt.legend(loc='lower left')
plt.ylim([0, 0.8])
plt.title(r'Elastic net performance as a function of the $\alpha$ parameter')
plt.xlabel(r'$\alpha$')
plt.ylabel(r'$R^2$');
```



We see that the optimal value of α is approximately 0.0005.

1.7.2 Exercise 6: (5 mts)

When choosing an optimal α, we split the data into two sets, train and test. The train set was used to fit the elastic net regularized linear regression model, while the test set was used to calculate the R-squared for each different value of α. What is a primary reason of splitting into train and test data when choosing an optimal α for this elastic net regression task?

(A) By choosing α using in-sample data, we reduce the risk of overfitting

(B) The model complexity is decreased which leads to a lower chance of overfitting

(C) To aide in out-of-sample model prediction accuracy, α is selected using the test data not involved in the fitting process

(D) Since α is a single variable to optimize, splitting into train and test reduces calculation time aiding in model development

Answer. (C). The explanation is given below.

Answer (A) is incorrect as we choose α using out-of-sample test data, not the in-sample train data that was used to fit the linear model.

Answer (B) is incorrect as the model complexity is not decreased by splitting into train and test sets. The model complexity remains unchanged, since the data the model is fit on or tested on is not involved in determining the complexity of the model.

Splitting into train and test data allows us to fit using the in-sample data, and then test on out-of-sample data to ensure the model does not overfit. Thus by selecting α using the testing data that is not involved in the modelling fitting (hence out-of-sample test data), this aides in having better out-of-sample prediction accuracy. Thus, answer (C) is the correct answer.

Though splitting into train and test data can indeed reduce training times in model development, this is not a reason why we split into training and testing data for choosing and optimal α. Thus, answer (D) is incorrect.

1.7.3 Exercise 7: (15 mts)

Write a script to change ρ and see how the optimal α changes (based on maximum test set R-squared as seen above). Namely, what are the optimal values of α using for the following values of ρ: [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]? Comment on what happens when ρ is 0 and when ρ is 1.

Answer. One possible solution is shown below:

[25]:
```
# Compare progression of train and test errors and alpha and rho vary
alphas = np.logspace(-5, 1, 50)
rhos = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
list_alpha_optim = []
for rho in rhos:
    train_errors = []
    test_errors = []
    enet = linear_model.ElasticNet(random_state=0, l1_ratio=0.5,␣
 ↪fit_intercept=True)
    for alpha in alphas:
        enet.set_params(alpha=alpha, l1_ratio=rho)
        enet.fit(X_train_simplified, y_train)
        train_errors.append(enet.score(X_train_simplified, y_train))
```

```
        test_errors.append(enet.score(X_test_simplified, y_test))

    i_alpha_optim = np.argmax(test_errors)
    alpha_optim = alphas[i_alpha_optim]
    print("Optimal alpha for rho = %.6f: %.6f" % (rho, alpha_optim))
    list_alpha_optim.append(alpha_optim)
```

```
Optimal alpha for rho = 0.100000: 0.002121
Optimal alpha for rho = 0.200000: 0.001207
Optimal alpha for rho = 0.300000: 0.000910
Optimal alpha for rho = 0.400000: 0.000687
Optimal alpha for rho = 0.500000: 0.000518
Optimal alpha for rho = 0.600000: 0.000518
Optimal alpha for rho = 0.700000: 0.000391
Optimal alpha for rho = 0.800000: 0.000391
Optimal alpha for rho = 0.900000: 0.000295
```

We find that the optimal alpha decreases as ρ increases.

1.8 Insights from the final elastic net model (5 mts)

For predicting home prices with a balanced portion of L1 and L2 regularization, we found an optimal $\alpha = 0.000518$ to use for the elastic net model. Let's take a look at our final model results and see if there are any insights we can draw:

[26]:
```
# Looking at the coefficents of the final model
final_elastic_net_model = linear_model.ElasticNet(random_state=0, alpha=0.
↪000518, l1_ratio=0.5, fit_intercept=True)
final_elastic_net_model.fit(X_train_simplified, y_train)
sorted(zip(X_train_simplified.columns, final_elastic_net_model.coef_),␣
↪key=lambda x: x[1])
```

[26]:
```
[('newConstruction', -0.010326892226584087),
 ('age', -0.005216691646158034),
 ('bedrooms', -0.004675685635988086),
 ('sewer_public/commercial', -0.0022462229233184395),
 ('heating_hot water/steam', -0.0015928929103743276),
 ('pctCollege', -0.0),
 ('fuel_electric', -0.0),
 ('sewer_septic', -0.0),
 ('fireplaces', 0.0005434853927543258),
 ('fuel_gas', 0.0019671783104647208),
 ('heating_hot air', 0.0030686697632882555),
 ('lotSize', 0.003513891569413089),
 ('centralAir', 0.004870730698973054),
 ('rooms', 0.006486034085846827),
 ('waterfront', 0.009525337302894227),
```

```
('bathrooms', 0.014121489970889516),
('landValue', 0.03345216803179957),
('livingArea', 0.04392189129273642)]
```

Since we chose to perform standardization of the data before fitting (to make the features on the same scale so as to not adversely influence the fit coefficents) we must take care in interpreting the output of the model.

The results of this elastic net model have pushed some of the coefficients toward zero, while fully reducing some of the less significant features to have zero coefficients (**fuel_electric**, **sewer_septic**, **pctCollege**).

We see that the housing prices are greatly positively influenced by **livingArea** and **landValue**. This follows our intuition as both of these quantities should naturally lead to a higher desired house price.

Moreover, the **newConstruction** and **age** variables have large negative betas. While age follows what we expect (older homes can have outdated features and thus can decline in price) the newConstruction feature may warrant further analysis as to why the variable leads to a negative coefficient. There may be some hidden variables at play that we lack data for; for example the new construction may be sourced by an untrusted contractor company which could lead to lower home prices.

1.9 Conclusions (5 mts)

In this case, we set out to determine salient variables that could help explain home prices. Through multiple linear regression, L1, L2, and elastic net regularization modeling we have found a number of useful and unhelpful variables for the analysis.

A number of variables that proved unhelpful include **fuel_oil** and **heating_electric**. Indeed, many of the categorical variables did not have significant p values so future analysis could investigate why this was the case.

Salient variables from the regression analysis included **landValue** and **livingArea**. Both of these variables showed a strong relationship to price in the original exploratory data analysis section, each had significant p values in the multiple linear regression section, and both were utilized through the L1, L2, and elastic net regularization sections.

1.10 Takeaways (5 mts)

In this case, we've applied the foundations of model regularization to housing price prediction. We covered L1 regularization, L2 regularization, overfitting and underfitting, elastic net regularization, and completed a series of multiple regression analyses.

Building on this knowledge, you can utilize regularization techniques to aid your models against overfitting and improve prediction accuracy. Robust predictions

are incredibly important not only for improved model estimates but also to help one maintain confidence in model predictions out-of-sample and in production settings. Quantitatively-driven business decisions depend on quality estimates, and regularization techniques are a useful tool to achieve that goal.