Lab 4

Introduction to Python Programming for Economics & Finance

Richard Foltyn
University of Glasgow

May 31, 2023

Contents

1 Predicting house prices with linear models

In this project, you will work with the Ames house data set which we already encountered in the lectures. Your task is to evaluate the following three linear models in terms of their performance when predicting house prices:

1. Linear regression
2. Ridge regression
3. Lasso

General hints:

1. Whenever a computation involves random number generation, initialise the seed to `123` to get reproducible results. Specifically, for `scikit-learn` functions this requires passing `random_state=123` where applicable.

1.1 Data description

The data is stored in `data/ames_houses.csv` in the course GitHub repository and can be downloaded using the link `https://raw.githubusercontent.com/richardfoltyn/python-intro-PGR/main/data/ames_houses.csv`.

To load the data, you need to specify the file path depending on your computing environment:

[1]:
```python
# Use this path to use the CSV file from the data/ directory
file = '../data/ames_houses.csv'
```

```
# Use this path if you want to download the file directly from Github
# file = 'https://raw.githubusercontent.com/richardfoltyn/python-intro-PGR/main/data/
↪ames_houses.csv'
```

You can load the CSV file as a pandas `DataFrame` as follows:

[2]:
```python
import pandas as pd

df = pd.read_csv(file, sep=',')

# Display columns in the data set
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   SalePrice         1460 non-null   float64
 1   LotArea           1460 non-null   float64
 2   Neighborhood      1460 non-null   object
 3   BuildingType      1386 non-null   object
 4   OverallQuality    1460 non-null   int64
 5   OverallCondition  1460 non-null   int64
 6   YearBuilt         1460 non-null   int64
 7   CentralAir        1460 non-null   object
 8   LivingArea        1460 non-null   float64
 9   Bathrooms         1460 non-null   int64
 10  Bedrooms          1460 non-null   int64
 11  Fireplaces        1460 non-null   int64
 12  HasGarage         1460 non-null   int64
dtypes: float64(3), int64(7), object(3)
memory usage: 148.4+ KB
```

The included variables are a simplified subset of the data available at openml.org:

- `SalePrice`: House price in US dollars (float)
- `LotArea`: Size of the lot in m^2 (float)
- `Neighborhood`: Name of the neighborhood (string)
- `BuildingType`: Type of building (categorical stored as string)
- `OverallQuality`: Rates the overall condition of the house from (1) "very poor" to (10) "excellent" (integer)
- `OverallCondition`: Rates the overall material and finish of the house from (1) "very poor" to (10) "excellent" (integer)
- `YearBuilt`: Original construction date (integer)
- `CentralAir`: Central air conditioning: Yes/No (categorical string)
- `LivingArea`: Above-ground living area in m^2 (float)
- `Bathrooms`: Number of bathrooms (integer)
- `Bedrooms`: Number of bedrooms (integer)
- `Fireplaces`: Number of fireplaces (integer)
- `HasGarage`: Indicator whether house has a garage (integer)

2 Data preprocessing

Apply the following steps to preprocess the data before estimation:

1. Drop all rows which contain any missing values (`NaN`)

   Hint: Use `dropna()` to remove rows with missing observations.

2. Recode the string values in column `CentralAir` into numbers such that `'N'` is mapped to 0 and `'Y'` is mapped to 1. Store this numerical variable using the column name `HasCentralAir`.

   Hint: You can use boolean operators such as `==` to create arrays containing `True` and `False`. You can then convert these to integer values 0 and 1 using `.astype(int)`:

   ```
   (df['CentralAir] == 'Y').astype(int)
   ```

3. Recode the values in column `Fireplaces` and create the new variable `HasFireplace` so that `HasFireplace = 1` whenever at least one fireplace is present and `HasFireplace = 0` otherwise.
4. Recode the string values in column `BuildingType` and create the new variable `IsSingleFamily` which takes on the value 1 whenever a house is a single-family home and 0 otherwise.
5. Convert the variables `SalePrice`, `LivingArea` and `LotArea` to (natural) logs. Name the transformed columns `logSalePrice`, `logLivingArea` and `logLotArea`.

3 Estimation

3.1 Model specification

You are now asked to estimate the following model of house prices as a function of house characteristics:

$$\begin{aligned}\log(SalePrice_i) = \alpha + f\Big(&\log(LivingArea_i),\ \log(LotArea_i),\ OverallCondition_i, \\ &OverallQuality_i,\ Bathrooms_i,\ Bedrooms_i\Big) \\ &+ \gamma_0 YearBuilt_i + \gamma_1 HasCentralAir_i + \gamma_2 HasFireplace_i + \gamma_3 IsSingleFamily_i + \epsilon_i\end{aligned}$$

where i indexes observations and ϵ is an additive error term. The function $f(\bullet)$ is a *polynomial of degree 3* in its arguments, i.e., it includes all terms and interactions of the given variables where the exponents sum to 3 or less:

$$\begin{aligned}f(\log(LivingArea_i), \log(LotArea_i), \dots) = {} & \beta_0 \log(LivingArea_i) + \beta_1 \log(LivingArea_i)^2 \\ & + \beta_2 \log(LivingArea_i)^3 + \beta_3 \log(LotArea_i) \\ & + \beta_4 \log(LotArea_i)^2 + \beta_5 \log(LotArea_i)^3 \\ & + \beta_6 \log(LivingArea_i) \log(LotArea_i) \\ & + \beta_7 \log(LivingArea_i)^2 \log(LotArea_i) \\ & + \beta_8 \log(LivingArea_i) \log(LotArea_i)^2 \\ & + \cdots\end{aligned}$$

Create a feature matrix `X` which contains all polynomial interactions as well as the remaining non-interacted variables.

Hints:

- Use the `PolynomialFeatures` transformation to create the polynomial terms and interactions from the columns `logLivingArea`, `logLotArea`, `OverallCondition`, `OverallQuality`, `Bathrooms` and `Bedrooms`.
- Make sure that the generated polynomial does *not* contain a constant ("bias"). You should include the intercept when estimating a model instead.
- You can use `np.hstack()` to concatenate two matrices (the polynomials and the remaining covariates) along the column dimension.
- The complete feature matrix `X` should contain a total of 87 columns (83 polynomial interactions and 4 non-polynomial features).

3.2 Train-test sample split

Split the data into a training and a test subset such that the training sample contains 70% of observations.

Hint:

- Use the function `train_test_split()` to split the sample. Pass the argument `random_state=123` to get reproducible results.
- Make sure to define the training and test samples only *once* so that they are identical for all estimators used below.

3.3 Linear regression

Perform the following tasks:

1. Do you need to standardise features before estimating a linear regression model? Does the linear regression model have any hyperparameters?
2. Estimate the above model specification using a linear regression model on the training sub-set.
3. Compute and report the mean squared error (MSE) on the test sample.

Hints:

- Use the `LinearRegression` class to estimate the model.
- The mean squared error can be computed with `mean_squared_error()`.

3.4 Ridge regression

Perform the following tasks:

1. Does Ridge regression require feature standardisation? If so, don't forget to apply it before fitting the model.
2. Use `RidgeCV` to determine the best regularisation strength α on the training sub-sample. You can use the MSE metric (the default) to find the optimal α. Report the optimal α and the corresponding MSE.
3. Plot the MSE (averaged over folds on the training sub-sample) against the regularisation strength α on the x-axis (use a log scale for the x-axis).
4. Compute and report the MSE on the test sample.

Hints:

- When running `RidgeCV`, use a grid of 500 α's which are spaced uniformly in logs: `python alphas = np.logspace(np.log10(1.0e-6), np.log10(100), 500)`
- Recall that the (negative!) best MSE is stored in the attribute `best_score_` after cross-validation is complete.

3.5 Lasso

Perform the following tasks:

1. Does Lasso require feature standardisation? If so, don't forget to apply it before fitting the model.
2. Use `LassoCV` to determine the best regularisation strength α on the training sub-sample using cross-validation with 5 folds. You can use the MSE metric (the default) to find the optimal α. Report the optimal α and the corresponding MSE.
3. Plot the MSE (averaged over folds on the training sub-sample) against the regularisation strength α on the x-axis (use a log scale for the x-axis).
4. Compute and report the MSE on the test sample for the model using the optimal α.
5. Report the number of non-zero coefficients for the model using the optimal α.

Hints:

- Getting Lasso to converge may require some experimentation. The following settings should help: increase the max. number of iterations to `max_iter=1000000` and use `selection='random'`. Set `random_state=123` to get reproducible results:

  ```
  LassoCV(..., max_iter=1000000, selection='random', random_state=123)
  ```

- Use `eps=1.0e-4` as an argument to `LassoCV` to specify the ratio of the smallest to the largest α.
- After cross-validation is complete, the MSE for each value of α and each fold are stored in the attribute `mse_path_` which is an array with shape `(N_ALPHA, N_FOLDS)`.

3.6 Compare estimation results

Create a table which contains the MSE computed on the test sample for all three models (using their optimal hyperparameters). Which model yields the lowest MSE?