2 REGRESSION AND REGULARIZATION

In this chapter, we discuss using (multiple) linear regression model to predict a target variable dependent on a number of features (or explanatory variables). We explain the use of non-overlapping data sets for training, validation, and testing for the purpose of prediction of future targets given future feature data. Such new future data outside of the training, validation, and test data sets are called generalized data.

The approach to prediction in machine learning typically adopts the following pathway. The data set is divided into a training data set (usually involving 50-60% of the data), a validation data set (usually involving 20-30% of the data), and a test data set (remaining part of the data set). The test data set is also called the hold-out data set. It is held out, and not involved in the training and validation phases. It is used only when the fitted model is ready for testing to check model usefulness to deploy in future with generalized data.

If there is no hyperparameter in the model/algorithm, there is no need for a validation data set; just the training data set (involving 60-80% of the data) and test data set (remaining part of the data set) will do.

A machine learning model or algorithm contains internal model parameters that have to be estimated or fitted to minimize the prediction or fitting errors between the targets (dependent variable) and the features (explanatory variables). The optimal fit is based on some specified criterion such as minimum mean square error (MMSE) in regression prediction or such as minimum cross entropy (MCE) in classification prediction. Hyperparameters are external to the model/algorithm in the sense that they are not explicitly estimated by using the data. They are exogenously specified and they control the estimation or fitting process and affects the parameter estimation and model fitting. If there is no hyperparameter in the model or algorithm, the model fitted using the training data set is then used to perform prediction based on features from the test data set. A fitted model means the model parameters and algorithmic structures are determined, and only input features need to be added to obtain the outputs that are predicted targets. The predicted targets or labels in the test data set are compared with the actual targets or labels in the test data set and the accuracy of the predictions is measured by some performance metric. If the latter performance is good, then the fitted model is ready for prediction in generalized data. If not, the model has to be re-considered or improved.

When there is a hyperparameter or there are hyperparameters in the model/algorithm, selection of optimal hyperparameter values ("fine-tuning" the hyperparameters) becomes an additional necessary step in the prediction process. In this situation, an additional non-overlapping validation data set is required.

Different hyperparameter(s) are chosen and each set of the hyperparameter values is used together with the training data set to find the optimized/fitted model. This fitted model (using training data set) is then applied to the validation data set features to obtain predicted targets. These predicted targets are then compared with the actual validation set targets to derive the accuracy measure. Optimal hyperparameters are found by maximizing this prediction accuracy measure. The optimal hyperparameters may be found via a search process or by trial and error. If the training set performance and validation set performance are similarly good, then one may proceed to employing the fitted model on the test data set.

A physical example may be described as follows. A robotic eye is trained to focus on a conveyor belt of manufactured widgets to predict the proportion of defects by learning the shape of the widget. The internal switches on the robotic eye were optimized or set in such a way that the first batch of training samples had prediction errors minimized, such as minimizing the percentage of errors. This was done using a particular speed of the conveyor belt. Using a second batch of validation samples, however, the prediction errors could be significantly larger.

The speed of the conveyor belt is a hyperparameter that may be altered and which will affect the learning of the widget shape parameters and thus the prediction accuracy of the robotic eye. By trying out different conveyor belt speeds, an optimal one could be chosen such that both training and validation samples yield good prediction accuracies or small prediction errors.

In this chapter we show the use of validation in a multiple linear regression model with regularization. Regularization is a method to constrain a model to fit out-of-sample (validation set) more accurately. Without regularization, the trained model may overfit and this can lead to underfitting or poor prediction performance out-of-sample. In linear regression, regularization takes the form of introducing constraints on the estimated coefficients to ensure they are not too large due to some significant impact of some features that may not occur uniformly or commonly across training and validation sample sets. Regularization is also useful in improving out-of-sample prediction when the

model is complex and there are too many features against a finite sample size or where there are high feature correlations that may sometimes produce higher prediction errors. Hyperparameters are used in regression with regularization.

The use of the validation set is to select the best model and associated best hyperparameter values so that the prediction performance in the validation data set is maximized. This result is compared with the training data set performance or in-sample fit on the same set of optimized hyperparameters to decide to proceed with the algorithm on the test data set provided both performances, especially the validation, are satisfactory. Validation also acts as a robustness check.

In a regression model, training is basically to find values of the regression coefficients or parameters that minimize prediction errors in some manner – this is fitting using training data. In the process of fitting, different selections of features may be tried to get the best fitting. Optimal or best fitting is accomplished typically in terms of a maximum fitting R^2 score or else minimum mean square error or least square error. R^2 is the coefficient of determination if it is the case of a linear regression. Maximizing R^2 is equivalent to minimizing a loss function.

After the model is trained or optimally fitted, it is used (using the fitted or estimated coefficients) on the test or hold-out data to check if the predicted target values are close to the actual target values in the test data set. This closeness is also commonly measured by a R^2 test score. If both training and test fittings are consistently good, then the model can be deployed for use in new data or generalized data not seen by the model in training or testing. Of course, in real-life applications, any model needs to be improved or re-trained frequently as more fresh data become available as updated training and test data.

Suppose the regression model or algorithm has hyperparameters. We will study many other algorithms in the rest of the book and will come across hyperparameters such as number of branches and depth in decision tree methods, number of nodes and layers in neural network methods, learning rate in gradient boosting methods, and so on. The training data can be used firstly to check if the predictive model or algorithm is feasible – if the in-sample prediction error is too large, e.g., wrong prediction more than 40% or 50% of the time or having a low R^2, given reasonable choices of

hyperparameters, then another predictive method or model should be considered.

If feasible, the model should next be checked for consistent performance as well as improved performance by optimal choice (fine-tuning) of the hyperparameters. This is done by a separate validation data set. In this context, the sample of size N could be divided into a training data set, e.g., 60% or 0.6N, a validation data set, e.g., 20% or 0.2N, and a test data set of the remaining 20% or 0.2N. For cross-sectional prediction, i.e., data not belonging to a time series, the sample points in the sets could be randomly assigned from the sample of size N. However, for time series, it is more suitable to assign sample points with the earliest time-stamps in the training set, followed by validation set, and the latest data in the test set. The validation data set prediction performance should return consistent performance of similar accuracy as in the training data. An inconsistent performance could be over-fitting or very high prediction accuracy in the training data set but under-fitting or high prediction error in the validation data set.

It is important to recall that fitting in the validation data set is not regressing validation data targets on validation data features. Optimal fitting in validation data set is using trained or fitted model from the training data set with selected hyperparameter(s) to predict validation data targets based just on the validation data features, and then to compare the fit of the actual target values against these predicted ones. If tuning the hyperparameters cannot eliminate the situation of high prediction error in the validation data set, then the predictive model could be problematic.

Suppose there are feasible competing models to perform the prediction of the target variable. Competing models are generally defined to be different algorithms or methods of prediction. These models use the same 0.6N training data set and the same 0.2N validation data set. Now there is the additional task of selecting the model amongst the competing models before testing (use of the test data set) is done. After fine-tuning the various models in the validation phase, the best performing model with the highest validation R^2 score is then selected for the testing. If the test data prediction returns similarly good performing results as the validation, then there would have been a reconfirmation, and the model would be suitable for use in a generalized data situation. If the testing returns poor performance such as substandard prediction, then the model has to be re-examined before applying it for generalized data prediction.

It is also possible that after the validation phase, an optional step could be to combine the original training and the validation data into a final training set, and this is used for a final training or fitting the coefficients (re-estimating the model parameters) with the larger sample size. The validation hyperparameters could be used for this purpose. The purpose here is not to change the number of features or to change or re-tune the hyperparameters, but to use the larger combined training data set to re-estimate and get better or less noisy coefficient estimates. This trained model is then used for checking the test R^2 in the test data set.

In this chapter, a useful method of cross-validation in machine learning is also discussed. This method allows validation to be conducted within the training data set itself and thus avoids use of another separate set of data for validation.

2.1 Regression and Regularization

Let Y_k be the k^{th} target variable observed and (X_{1k} , X_{2k} , X_{3k} , ….. , X_{pk}) be the set of p features related to the k^{th} target variable. The linear regression is written as

$$Y_k = b_0 + b_1 X_{1k} + b_2 X_{2k} + b_3 X_{3k} + \ldots\ldots + b_p X_{pk} + \varepsilon_k \qquad (2.1)$$

where ε_k is a residual variable that is unobserved. Unlike some classical setup in linear regression where the added residual noise or innovation may take specific distributional assumptions to enable testing, the residual noise in the machine learning method takes a lower profile. We shall employ several competing models in the regression of type in Eq. (2.1). Therefore, we use 0.6N amount of data points for training, 0.2N amount for validation, and the remaining 0.2N for testing.

The linear regression under supervised machine learning in Eq. (2.1) typically aims to find optimal parameter (estimates) $\hat{b}_0$, $\hat{b}_1$, $\hat{b}_2$, …., $\hat{b}_p$ so that a loss criterion such as sum of squared errors, $\sum_{k=1}^{0.6N}(Y_k - \hat{Y}_k)^2$ is minimized, where fitted $\hat{Y}_k = \hat{b}_0 + \hat{b}_1 X_{1k} + \hat{b}_2 X_{2k} + \cdots + \hat{b}_p X_{pk}$. Minimizing sum of squared errors is the same as finding estimates with MMSE $\frac{1}{0.6N}\sum_{k=1}^{0.6N}(Y_k - \hat{Y}_k)^2$. The R^2-score for the training data set is 1 – $\sum_{k=1}^{0.6N}(Y_k - \hat{Y}_k)^2 / \sum_{k=1}^{0.6N}(Y_k - \overline{Y}_k)^2$. The second term is sum of squared errors

over total sum of squares in standard least squares regression terminology, and R^2 lies between 0 and +1.

When the trained or fitted coefficients $\hat{b}_0$, $\hat{b}_1$, $\hat{b}_2$, …., $\hat{b}_p$ are employed in the prediction of targets Y_k's in the validation data set using the associated validation set data of X_{jk}'s (for j =1,2,…,p), the prediction yields $Y_k^* = \hat{b}_0 + \hat{b}_1 X_{1k} + \hat{b}_2 X_{2k} + \cdots + \hat{b}_p X_{pk}$. The prediction validation score is $R^2 = 1 - \sum_{k=1}^{0.2N}(Y_k - Y_k^*)^2 / \sum_{k=1}^{0.2N}(Y_k - \overline{Y}_k)^2$. In this case, the score R^2 may be sometimes negative if the actual model of the validation data is different so that $\sum_{k=1}^{0.2N}(Y_k - Y_k^*)^2$ is very large. The R^2 score in the validation set is a useful measure of accuracy of the machine learning method when coefficients (or model parameters) trained by a training data set is applied to new data. A somewhat similar though not identical metric in measuring prediction performance is the root mean square error (RMSE) which is $\sqrt{\sum_{k=1}^{0.2N}\left(Y_k - Y_k^*\right)^2/(0.2N)}$.

The validation performance will not be good if the assumption that both data sets come from the same data generating model is incorrect. The use of training and validation data sets is akin to the idea of in-sample and out-of-sample fits. The above comments on the validation data set apply similarly to use of trained or fitted model in prediction in the test data set.

It should be noted that where $\{Y_k, X_{1k}, X_{2k}, X_{3k}, \ldots\ldots, X_{pk}\}_{k=1,2,\ldots N}$ is not a time series, but a cross-section, then splitting the data set into training, validation, and test subsets can be random. However, if it is a time series, then typically the training set data would precede those of the validation data set. Both should precede that of the test set. Moreover, in the strict sense of prediction based on available information, the features would have time-stamps prior to that of the target variable.

If a training set R^2 or accuracy of fit by the model is high and if a validation set R^2 score is also high, then the fitted model with its fine-tuned hyperparameters is used to check out the test data set. If the test R^2 score is also high or having accurate prediction based on the trained or fitted coefficients, then the model with its trained coefficients may be usable for future prediction of the target variable when a new set of features arrives.

Regressions based on available features can run into overfitting problem when too many features (or explanatory variables) are utilized. This may produce good fit, e.g., high R^2 in the training data set, but the fitted coefficients

may in turn produce inaccurate predictions using the validation and the test data sets.

Consider the multivariate regression (2.1) where we re-write in short-form its predicted value as $\hat{Y}_k = \sum_j \hat{b}_j X_{jk}$. The sum of squared prediction errors is then $\Sigma_k \left(Y_k - \Sigma_j \hat{b}_j X_{jk}\right)^2$. Suppose an extra feature Z_k for instance or sample point k is added. Then the sum of squared prediction errors is now $\Sigma_k \left(Y_k - \Sigma_j \hat{b}_j X_{jk} - \hat{b}_{p+1} Z_k\right)^2$. Clearly this latter SSE is smaller, i.e.,

$$\Sigma_k \left(Y_k - \Sigma_j \hat{b}_j X_{jk} - \hat{b}_{p+1} Z_k\right)^2 \leq \Sigma_k \left(Y_k - \Sigma_j \hat{b}_j X_{jk}\right)^2$$

as the MMSE on the left-hand term is at most as large as the right-hand term when we put $\hat{b}_{p+1} = 0$. Hence unnecessarily increasing the number of features that do not actually affect the target can produce overfitting (higher in-sample training data set R^2). When a validation or test set is used however, there will be underfitting as the additional irrelevant features can distort the prediction and lead to low validation or test score.

The training data set may contain some large outlier instances or sample points that may not occur uniformly or commonly across training and validation sample sets. These kinds of outliers may not occur in the validation or test data sets. By trying to fit with MMSE on training data set with these outliers (that are not removed or not examined), coefficients associated with these outlier features may be fitted with values of larger magnitudes than if the outliers did not occur. This creates overfitting in the training data set (higher R^2) relative to the validation or test set that will see underfitting (lower R^2) when the fitted coefficients are used for prediction (with same hyperparameters).

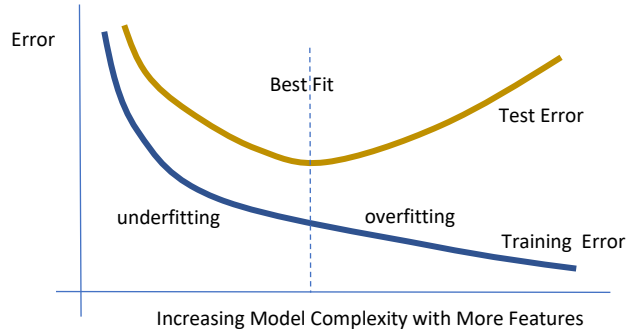
Overfitting could also occur when some features are highly multi-collinear, i.e., having high correlations. We illustrate as follows. Suppose a correctly specified regression is $Y_k = b_0+b_1X_k+\varepsilon_k$ where Y=(5,5.1,4.6,4.9). X=(9,7,8.5, 7.5). A simple regression yields R^2 of 0.18. Consider another feature Z that is highly correlated with X. Z=(1,3,0.5,2.5). Their correlation is –0.92. If we add Z to the regression, i.e., $Y_k = b_0+b_1X_k+b_2Z_k$+residual error, the R^2 becomes 0.87. When X and Z are added in the regression, their negative correlation smooths out their combined fluctuations, producing aggregated $0.36X_k+0.38Z_k$ that correlated highly with Y and hence yields a lower R^2.

The high multi-collinearity could produce overfitting in the training data set but in a different sample such as validation or test data set, the realizations

of X and Z in small sample may not follow the correlation pattern exactly. In such a case, it is more likely that since the correct specification is without Z, the out-of-sample prediction would be poorer with underfitting. Other approaches of regression avoiding poor out-of-sample fit due to multi-collinearity include selection of features, stepwise regression, and principal component regression or regression using key principal components.

Underfitting occurs when the fitted R^2's or the prediction R^2 are low. Obviously, if training R^2 is too low after optimization, the algorithm is ineffective and should be changed – there is no need to proceed to testing. With increasing number of features, it is possible that the training R^2 will increase and equivalently its training error (this could be defined as $1 - R^2$) will decrease. However, the validation R^2 score and the test R^2 score could still be low.

There is possibly an ideal balance between overfitting and underfitting for the training data as seen in the graph. At the ideal balance, it is likely that the validation and the test data fits will be best (least error). This is seen in the graph below.



To reduce overfitting and enable good validation and test data predictions, besides feature selections, some constraints can be added to the regression. A penalty function is added to reduce the tendencies to yield large-fitted coefficients – this technique is called regularization. There are three common regularized linear regression methods – Ridge regression, Lasso (Least absolute shrinkage and selection operator) regression, and Elastic Net regression. Regularization could shrink some fitted coefficients or even reduce them toward zero, effectively reducing the dimension space of the features

since those affected features become impactless if their fitted coefficients are close to zero. They are also called shrinkage estimators. They reduce the ill effects of excessive number of features, outsized outliers and also high multi-collinearity.

For Ridge regression, see the module sklearn.linear_model.Ridge with documentation in https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html. One can also check this source for the explicit codes. This module is the Ridge regression model that minimizes the objective function

$$\sum_{k=1}^{n}\left(Y_k - \sum_{j=0}^{p} b_j X_{jk}\right)^2 + \alpha\left(\sum_{j=1}^{p} b_j^2\right) \qquad (2.2)$$

where $X_{0k} = 1$. The second term is in L^2-norm (sum of squared values of coefficients – related to measure of Euclidean distance), so Eq. (2.2) is also called a L^2 regularized regression. Note there is no constraint on the intercept. When we use the minimization in sklearn linear_model.Ridge ( ), the default parameters within ( ) include setting alpha > 0, e.g., $\alpha = 0.1$. We can of course try different values for the alpha hyperparameter to attain low training error.

For Lasso regression using the module sklearn.linear_model.Lasso, see documentation in https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html. One can also check this source for the explicit codes. This module is the Lasso regression model that minimizes the objective function

$$\sum_{k=1}^{n}\left(Y_k - \sum_{j=0}^{p} b_j X_{jk}\right)^2 + \alpha \sum_{j=1}^{p}|b_j| \qquad (2.3)$$

where $X_{0k} = 1$. The second term is in L^1-norm (sum of absolute values of coefficients), so Eq. (2.3) is also called a L^1 regularized regression. When we use the minimization in sklearn linear_model.Lasso ( ), the default parameters within ( ) include setting alpha > 0, e.g., $\alpha = 0.1$. We can of course try different values for the alpha hyperparameter to attain low training error.

Another possible regularized regression is the 'elastic net' regression that is the minimization of objective function as follows, where there are two hyperparameters α_1 and α_2.

$$\sum_{k=1}^{n}\left(Y_k - \sum_{j=0}^{p} b_j X_{jk}\right)^2 + \alpha_1\left(\sum_{j=1}^{p} b_j^2\right) + \alpha_2 \sum_{j=1}^{p}|b_j| \qquad (2.4)$$

This is a combination of regularizations from the Ridge and also the Lasso methods.

2.2 Worked Example – Data

We explain the regularized regressions based on a study to predict Singapore Housing Development Board (HDB) resale flat (house) prices. See the publicly accessible data sets at https://data.gov.sg/collections/189/view. This method and other related methods in machine learning can be applied to predicting housing prices anywhere as long as there are adequate data pertaining to the house prices and features of each house. See demonstration file Chapter2-1.ipynb.

We use a data set that comprises only 13 features (characteristics). 62,359 samples (sample points) are available in the data set HDBflat_value.csv, each of which is a flat in Singapore. The data cover the period 2017 through to 2023. The target variable is 'resale_price_persqm' – the transacted resale price of a 4-room flat in terms of Singapore dollars per square metre of the flat floor area.

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[2]:
```
df = pd.read_csv('HDBflat_value.csv')
### Historical data of Singapore HDB Resale prices
### are available at https://data.gov.sg/collections/189/view
df.head()
```

[3]:
```
df = df.drop(columns=['flat_type', 'resale_price', 'persqm_price_exceed_6500'])
### delete 'district_E' and 'Simplified' columns as they create col dependencies
df = df.drop(columns=['district_E', 'Simplified'])
df.isnull().sum()
### this is to check if there is any missing data in the index
###  and other columns -- none or '0'
```

[4]:
```
### Convert 'year' to no. of years after 2016
df['year'] = df['year'] - 2016
```

[5]:
```
df=df.iloc[:,0:13]
df.head()
```

[5]:

	year	height	floor_area_sqm	remaining_lease	district_C	district_N	district_S	district_W	Improved	Model_A	New_Gen	Premium	resale_price_persqm
0	1	3	82	63.500000	1	0	0	0	1	0	0	0	4695.121951
1	1	6	83	61.666667	1	0	0	0	1	0	0	0	7469.879518
2	1	6	85	63.583333	1	0	0	0	1	0	0	0	4705.882353

The features, after some pre-processing, are:

(1) ‘year’ – number of years after 2016 in which the resale took place, with resale in 2017 indicated as ‘1’, resale in 2018 indicated as ‘2’, and so on.
(2) ‘height’ – the level of the flat. For example, ‘9’ indicates that the flat was in the 9th storey.
(3) ‘floor_area_sqm’ – the floor area of the flat
(4) ‘remaining_lease’ – the number of years left in the lease. All Singapore government flats have a 99 years old lease at the start. By the end of the lease, the flat reverts back to the government.
(5) ‘district_C’ – location of the flat in the central region of Singapore
(6) ‘district_N’ – location of the flat in the northern region of Singapore
(7) ‘district_S’ – location of the flat in the southern region of Singapore
(8) ‘district_W’ – location of the flat in the western region of Singapore
(9) ‘Improved’ – a specific design of the 4-room flat
(10) ‘Model_A’ – a specific design of the 4-room flat
(11) ‘New_Gen’ – a specific design of the 4-room flat
(12) ‘Premium’ – a specific design of the 4-room flat

Command “df.describe()” in Pandas provides summary descriptive statistics of column variables in the DataFrame. These include mean, standard deviation, minimum, maximum, and the various percentiles for the numerical data.

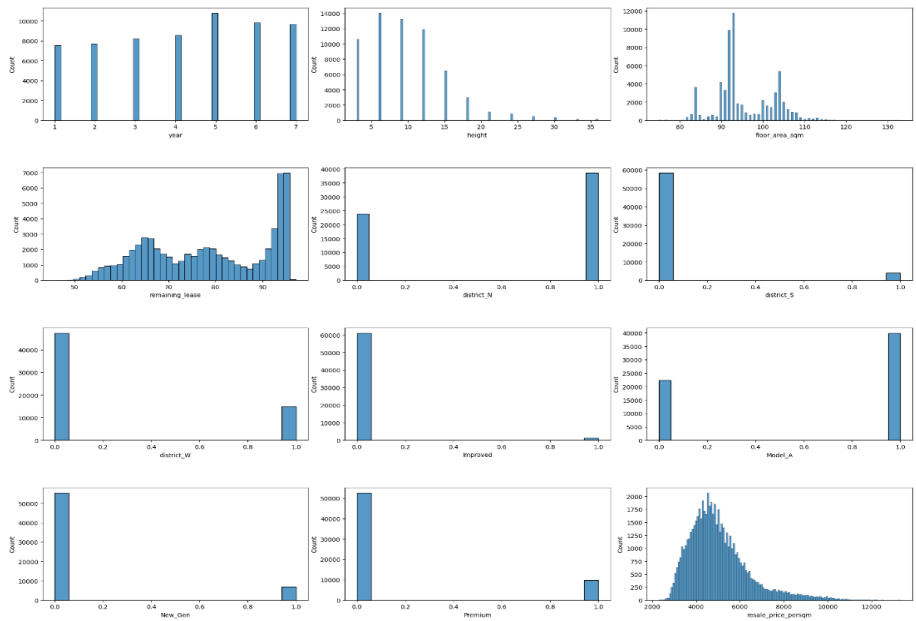
```
[7]: # Let's summarize the data to see the distribution of data
     print(df.describe())
     ### 62,359 rows of data or cases from year 2017 thru 2023 (may not
     ###  be complete data collected here). 'flat_type' all 4-Room.
```

```
                year        height  floor_area_sqm  remaining_lease  \
count  62359.000000  62359.000000    62359.000000     62359.000000
mean       4.214516      9.757469       95.475232        78.488792
std        1.968071      5.589792        6.857665        13.050042
min        1.000000      3.000000       75.000000        45.750000
25%        3.000000      6.000000       92.000000        66.416667
50%        4.000000      9.000000       93.000000        78.500000
75%        6.000000     12.000000      102.000000        92.666667
max        7.000000     36.000000      133.000000        97.166667

         district_C    district_N    district_S    district_W      Improved  \
count  62359.000000  62359.000000  62359.000000  62359.000000  62359.000000
mean       0.004169      0.617393      0.064289      0.238618      0.020478
std        0.064437      0.486027      0.245269      0.426242      0.141630
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%        0.000000      0.000000      0.000000      0.000000      0.000000
50%        0.000000      1.000000      0.000000      0.000000      0.000000
75%        0.000000      1.000000      0.000000      0.000000      0.000000
max        1.000000      1.000000      1.000000      1.000000      1.000000
```

	Model_A	New_Gen	Premium	resale_price_persqm
count	62359.000000	62359.000000	62359.000000	62359.000000
mean	0.640870	0.111708	0.157058	5027.828256
std	0.479749	0.315009	0.363859	1375.536497
min	0.000000	0.000000	0.000000	2307.692308
25%	0.000000	0.000000	0.000000	4086.021505
50%	1.000000	0.000000	0.000000	4772.277228
75%	1.000000	0.000000	0.000000	5652.173913
max	1.000000	1.000000	1.000000	13310.344830

Code line [8] explores the empirical distributions of each feature. Such preliminary explorations provide some ideas of which features may be more important in the prediction.

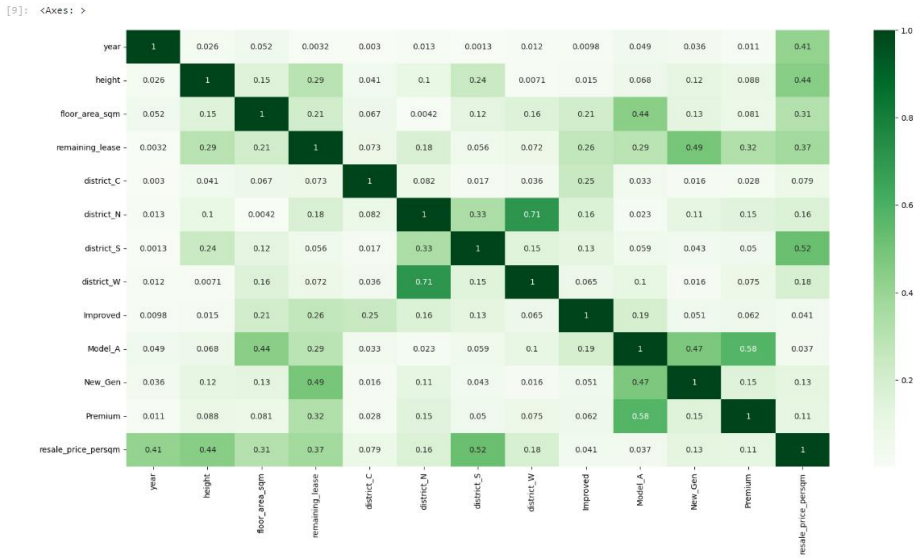
```
[8]: ### Proportion of '1''s in 'district_C' is very small, so we left out of graph
df1 = df.drop(columns=['district_C'])

import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats

fig, axs = plt.subplots(ncols=3, nrows=4, figsize=(20, 15))
index = 0
axs = axs.flatten()
for k,v in df1.items():
   sns.histplot(v, ax=axs[index])
   index += 1
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=5.0)
### uses seaborn
```



Code line [9] shows the correlation heat map amongst the features, including the target variable.

```
[9]: ### pairwise correlation on the features including 'resale_price_persqm'
     plt.figure(figsize=(20, 10))
     sns.heatmap(df.corr().abs(), annot=True, cmap="Greens" )
     ### In heat map, clearly tax and rad are highly correlated
     ### For more options - see https://seaborn.pydata.org/generated
     ###  /seaborn.heatmap.html
```

In the last row, it is seen that ‘year’, ‘height’, ‘remaining_lease’, and location ‘district_S’ all have high positive correlations with the target variable ‘resale_price_persqm’.



Next we define the target variable y and also the features X. The vector y and matrix X are then divided into (60%) rows forming corresponding X_train, y_train in training set, (20%) rows forming corresponding X_val, y_val in the validation set, and the remaining (20%) rows forming X_test, y_test in the test set. The rows are randomly assigned to the separate data sets via a random_state. When formed, the training, validation, and test data sets have correspondingly 37,415, 12,472, and 12, 472 sample sizes.

[10]:
```
### Here we define the target var y and the features X
X = df.loc[:,['year','height','floor_area_sqm','remaining_lease',\
              'district_C','district_N','district_S','district_W',\
              'Improved','Model_A','New_Gen','Premium']]
y = df['resale_price_persqm']
```

[11]:
```
### Splitting dataset into Training set 60%, Validation Set 20% and Test set 20%
from sklearn.model_selection import train_test_split
X_train, X_valtest, y_train, y_valtest = train_test_split(X, y, test_size = 0.4, \
                                                  random_state = 37)
### X_valtest and y_valtest contains 20% validation and 20% test data
X_val, X_test, y_val, y_test = train_test_split(X_valtest, y_valtest, test_size = 0.5, \
                                                  random_state = 14)
len(X_train), len(y_train), len(X_val), len(y_val), len(X_test), len(y_test)
```

[11]:
```
(37415, 37415, 12472, 12472, 12472, 12472)
```

2.3 Linear Regression Prediction

Continuing with the worked example, a linear regression model is used as a base comparison model to predict the flat resale prices (per square metre) in subsection 2.2. For comparison with the regularized models with hyperparameters, we employ the training set (60% data) for the initial fitting. This is done in code lines [13] and [14].

[13]:
```
### see module documentation in https://scikit-learn.org/stable/modules
###  /generated/sklearn.linear_model.LinearRegression.html
from sklearn.linear_model import LinearRegression
Linreg = LinearRegression()
Linreg.fit(X_train, y_train)
### To retrieve the intercept:
print('Intercept (train):', Linreg.intercept_)
### To retrieving the slope:
print('Slopes (train):', Linreg.coef_)

from sklearn.metrics import r2_score
y_pred_Linreg_train = Linreg.predict(X_train)
### 'y_pred_Linreg_train' is predicted y using the x_train data
r2_score_Linreg_train = r2_score(y_train, y_pred_Linreg_train)
### r2_score_Linreg_train is R-sq in lin reg involving training data set
print('Linreg_train_R2_score: ', r2_score_Linreg_train)
```

```
Intercept (train): 3422.055996603847
Slopes (train): [  281.167182      53.05749049   -19.83328235    32.31417559
  1295.99801638  -972.2729605   1601.62306092 -1088.95040275
  -261.64604368   -18.87466348   125.04386431    94.53231616]
Linreg_train_R2_score:  0.6721950473196396
```

[14]:
```
from sklearn.metrics import mean_squared_error
### Predicting RMSE -- the Training set results
rmse_Linreg_train = (np.sqrt(mean_squared_error(y_train, y_pred_Linreg_train)))
print("Linreg_train_RMSE: ", rmse_Linreg_train)
```

```
Linreg_train_RMSE:  786.2624684200345
```

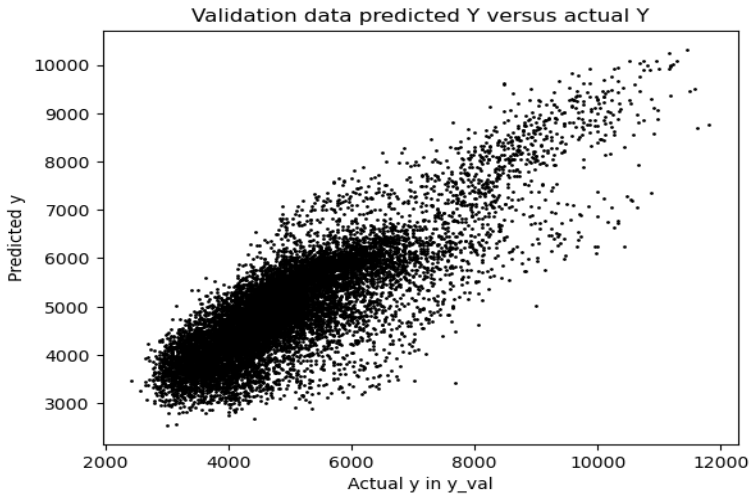
In the fitting using the 60% training data set, the fitted R^2 (or R^2 score) is 0.672195 with RMSE of 786.2625.

In code line [16], the fitted model is then used to predict targets in the validation data set employing features from the validation data set. The linear regression validation R^2 score is 0.69452567 with RMSE 769.1847. It is possible, as in this case, that the prediction outcomes are better than in the fitted model.

```
[16]: ### Predicting R2 Score using Validation Set but Intercept(train)
      ###  and Slopes(train) from Training results
      y_pred_Linreg_val = Linreg.intercept_ + np.dot(X_val,Linreg.coef_.T)

      r2_score_Linreg_val = r2_score(y_val, y_pred_Linreg_val)
      ### r2_score_Linreg_val is R-sq in lin reg involving validation data set
      print('Linreg_val_R2_score: ', r2_score_Linreg_val)

      rmse_Linreg_val = (np.sqrt(mean_squared_error(y_val, y_pred_Linreg_val)))
      print("Linreg_val_RMSE:",rmse_Linreg_val)

      Linreg_val_R2_score:  0.6945256718342248
      Linreg_val_RMSE: 769.1847041142491
```

The predicted target in the validation set is plotted against the actual target values and this is shown in code line [18].

```
[18]: ### A scatterplot may be clearer - here matplotlib
      plt.scatter(y_val, y_pred_Linreg_val,s=1,color='black')
      plt.title("Validation data predicted Y versus actual Y")
      plt.xlabel("Actual y in y_val")
      plt.ylabel("Predicted y")
      plt.show()
```



2.4 Fine-Tuning Hyperparameters

The Ridge, Lasso, and Elastic Net regressions with constraints as in Eqs. (2.2), (2.3), and (2.4) are then used to fit the data in the training set. For each model in (2.2), (2.3), and (2.4), different hyperparameter value(s) each time is employed to fit the model using the training set data. The fitted model is then used to perform prediction based on features from the validation set. The validation prediction R^2 score is noted, and the hyperparameter values are iterated till a maximum validation R^2 score is obtained. The optimal (best or "fine-tuned") hyperparameter value(s) and the associated maximum validation R^2 score are noted for each model.

Code line [21] shows the outcome for the Ridge regression model.

Ridge

[21]:
```
from sklearn import linear_model
from sklearn.metrics import r2_score

base = 0.6

num = 0.0
while num < 5.0:

    Ridge=linear_model.Ridge(alpha=num,)
    Ridge.fit(X_train, y_train)

    y_pred_Ridge_val = Ridge.intercept_ + np.dot(X_val,Ridge.coef_.T)
    r2_score_Ridge_val = r2_score(y_val, y_pred_Ridge_val)
    rmse_y_pred_Ridge_val = (np.sqrt(mean_squared_error(y_val, y_pred_Ridge_val)))

    if r2_score_Ridge_val > base:
        base = r2_score_Ridge_val
        alpha_Ridge = num
        RI = Ridge.intercept_
        RC = Ridge.coef_
        RMSE = rmse_y_pred_Ridge_val
    num = num + 0.01

print("OPTIMAL HYPERPARAMETER in VALIDATION SCORE")
print("alpha:", alpha_Ridge)
print('Intercept (train):', RI)
print('Slopes (train):', RC)
print("r2_score_Ridge_validation:", base)
print("Pred_RMSE_Ridge_validation:", RMSE)
```

```
OPTIMAL HYPERPARAMETER in VALIDATION SCORE
alpha: 3.5399999999999685
Intercept (train): 3425.557792489438
Slopes (train): [  281.13954996    53.10300082   -19.86813128    32.30366725
  1262.57317609  -972.43971792  1597.96984447 -1088.88032408
  -255.83874467   -18.01989755   125.29255267    95.05039738]
r2_score_Ridge_validation: 0.6945282132520314
Pred_RMSE_Ridge_validation: 769.1815044611373
```

Optimal alpha with highest R^2 score is 3.54. Compared with the base regression model with no constraint, the validation R^2 score is marginally higher at 0.69452821 while the RMSE is marginally lower at 769.1815. The estimated coefficients for features 'district_C', 'district_S', and 'Improved' also have smaller magnitudes due to the Ridge regression constraint. Hence the constraints lead to better prediction.

It should be noted that while it is convenient to employ an optimization package like Sklearn for the regressions, they use objective functions that are scaled differently to facilitate numerical solutions. For example, sklearn uses min $\| y - Xw \|^2 + \alpha \|w\|^2$ for Ridge regression just as in Eq. (2.2), with vector w consisting of the parameters b_k's as elements. But sklearn uses min $(2n)^{-1} \| y - Xw \|^2 + \alpha \|w\|$ for Lasso regression, which is different from Eq. (2.3) as the alpha in Sklearn Lasso has to be scaled up by (2n) to make it equivalent to the alpha in (2.3), where n is the sample size in the regression. And in sklearn's Elastic Net regression, the object is min $(2n)^{-1} \| y - Xw \|^2 + 0.5 \times \alpha \times (1 - \text{l1\_ratio}) \times \|w\|^2 + \alpha \times \text{l1\_ratio} \times \|w\|$, which has different scales as in Eq. (2.4). In the latter, comparing with Eq. (2.4), $\alpha_1 \equiv n \times \alpha \times (1 - \text{l1\_ratio})$ and $\alpha_2 \equiv 2n \times \alpha \times \text{l1\_ratio}$, where α and l1_ratio are input values to the sklearn optimization.

l1_ratio is called the ElasticNet mixing parameter. If l1_ratio = 1, the penalty or constraint would just be the L1 penalty or just the Lasso regression. If l1_ratio = 0, the penalty or constraint would just be the L2 penalty or just the Ridge regression. The computable range of l1_ratio is between 0 and 1.0 inclusive. If the mixing parameter lies within (0,1), it would be a combination of L1 and L2 constraints.

Thus, for the various regularized regressions, we would not venture into troublesome comparisons of the hyperparameter values across different models. For each model, just as in the Ridge regression above, the optimal or "fine-tuned" hyperparameter is found which led to the maximum R^2 scores.

The latter is what is important for comparisons across the models for their validation data set performances.

Code line [23] shows the outcome for the Lasso regression model.

Lasso

[23]:
```
from sklearn import linear_model
from sklearn.metrics import r2_score

base = 0.6

num = 0.0
while num < 0.3:

    Lasso=linear_model.Lasso(alpha=num,)
    Lasso.fit(X_train, y_train)

    y_pred_Lasso_val = Lasso.intercept_ + np.dot(X_val,Lasso.coef_.T)
    r2_score_Lasso_val = r2_score(y_val, y_pred_Lasso_val)
    rmse_y_pred_Lasso_val = (np.sqrt(mean_squared_error(y_val, y_pred_Lasso_val)))

    if r2_score_Lasso_val > base:
        base = r2_score_Lasso_val
        alpha_Lasso = num
        LI = Lasso.intercept_
        LC = Lasso.coef_
        RMSE = rmse_y_pred_Lasso_val
    num = num + 0.001

print("OPTIMAL HYPERPARAMETER in VALIDATION SCORE")
print("alpha:", alpha_Lasso)
print('Intercept (train):', LI)
print('Slopes (train):', LC)
print("r2_score_Lasso_validation:", base)
print("Pred_RMSE_Lasso_validation:", RMSE)
```

```
OPTIMAL HYPERPARAMETER in VALIDATION SCORE
alpha: 0.04300000000000003
Intercept (train): 3420.4820939032215
Slopes (train): [  281.15356589    53.07079852   -19.8268077     32.32253673
  1283.35605292  -971.77259527  1600.85485195 -1088.3703868
  -257.48300225   -18.99733798   124.73335914    93.93814603]
r2_score_Lasso_validation: 0.6945260996464644
Pred_RMSE_Lasso_validation: 769.1841654982089
```

Optimal alpha is 0.043. Compared with the base regression model with no constraint, the validation R^2 score is marginally higher at 0.69452610 while the RMSE is marginally lower at 769.1841. The performance, however, is not as good as that of the Ridge regression.

Code line [25] shows the outcome for the Elastic Net regression model.

Elastic Net

[25]:
```
from sklearn import linear_model
from sklearn.metrics import r2_score

base = 0.6

num1 = 0.0
while num1 < 0.1:

    num2 = 0.0
    while num2 < 1.0:

        ELNet=linear_model.ElasticNet(alpha=num1,l1_ratio=num2)
        ELNet.fit(X_train, y_train)

        y_pred_ELNet_val = ELNet.intercept_ + np.dot(X_val,ELNet.coef_.T)
        r2_score_ELNet_val = r2_score(y_val, y_pred_ELNet_val)
        rmse_y_pred_ELNet_val = (np.sqrt(mean_squared_error(y_val, y_pred_ELNet_val)))

        if r2_score_ELNet_val > base:
            base = r2_score_ELNet_val
            alpha_EN = num1
            l1_ratio_EN = num2
            EI = ELNet.intercept_
            EC = ELNet.coef_
            RMSE = rmse_y_pred_ELNet_val

        num2 = num2 + 0.1
    num1 = num1 + 0.001

print("OPTIMAL HYPERPARAMETER in VALIDATION SCORE")
print("alpha:", alpha_EN)
print("l1_ratio:", l1_ratio_EN)
print('Intercept (train):', EI)
print('Slopes (train):', EC)
print("r2_score_ELNet_validation:", base)
print("Pred_RMSE_ELNet_validation:", RMSE)
```

```
OPTIMAL HYPERPARAMETER in VALIDATION SCORE
alpha: 0.001
l1_ratio: 0.8999999999999999
Intercept (train): 3425.7175670985744
Slopes (train): [  281.13772558    53.10583882   -19.86993016    32.30326366
  1260.46508037  -972.436558    1597.74950211 -1088.86195051
  -255.4296934    -17.97579224   125.29936726    95.06568431]
r2_score_ELNet_validation: 0.6945281801665097
Pred_RMSE_ELNet_validation: 769.1815461159999
```

Optimal alpha is 0.001 and optimal l1_ratio is 0.899. The validation R^2 score is 0.69452818 while the RMSE is marginally lower at 769.1815. The performance is just marginally lower than that of the Ridge regression.

The validation R^2 scores for the linear regression with no constraints, the Ridge, Lasso, and Elastic Net regressions are tabulated below with the optimal hyperparameters shown in the brackets.

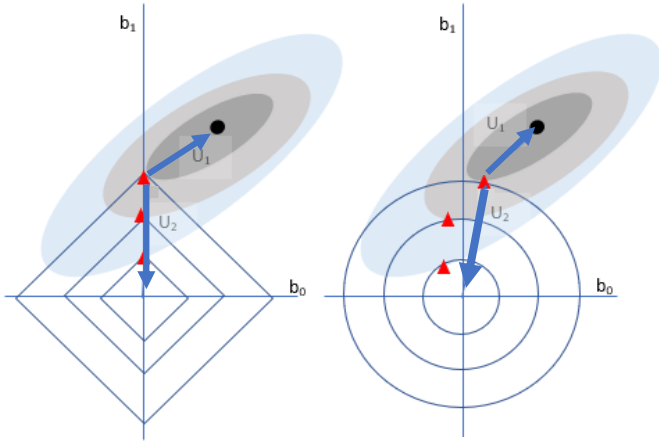
	Validation R^2
Linear Regression (no constraints)	0.69452567
Ridge Regression (sklearn α = 3.54)	0.69452821
Lasso Regression (sklearn α = 0.043)	0.69452610
Elastic Net Regression (sklearn α = 0.001, l1_ratio = 0.899)	0.69452818

In the data studied, the various regressions produce close solutions. In general, however, there are some differences between the Ridge and the Lasso regression due to the difference in their constraints. Their difference can be illustrated geometrically as follows using a two-dimensional case of minimizing the sum of squared errors

$$\sum_{k=1}^{n} (Y_k - b_0 - b_1 X_{1k})^2$$

that is geometrically an ellipse centred at (b_0, b_1), taking the values (Y_k, X_k)'s as given.

Under regularization, its minimization is subject to the Lasso type constraints (LHS below) and the Ridge type constraints (RHS below).



Lasso constraints

Ridge constraints

The contoured rings, contoured squares, and contoured circles represent equal levels.

It can be seen in the diagrams that the black dot is the (b_0*,b_1*) optimal point for the minimization of the unconstrained least squares regression. U_1 denotes adjusting the (b_0,b_1) estimates toward minimizing the unconstrained objective function $\sum_{k=1}^{n}\left(Y_k - \sum_{j=0}^{p} b_j X_{jk}\right)^2$ while U_2 denotes pulling solution (b_0,b_1) toward minimizing the penalty function. Minimizing the combination of the quadratic function and the penalty constraint is like pulling in opposite directions and the minimum is reached at a point when both contours meet tangentially at points marked as triangles. Higher alphas will pull the solution toward the origin and vice-versa. For a given alpha, the minimum solution will not be away from the tangential point as the sum of the two components will then be larger. It is seen that for Lasso, often the optimal points involve zeros in some coefficients.

In general, for problems with a much larger set of features, Lasso models are adept at feature selection – i.e., selecting a reduced/smaller set of features from a large set. This is due to the large gradient in the penalty function– a feature selection characteristic of the lasso penalty when the coefficients are less than one. An outcome is that typically when two strongly correlated features are pushed towards zero, one may be eliminated. It may therefore ignore or remove some features that may, nevertheless, be interesting or important. In contrast, the ridge regression penalty reduces correlated features together without quickly removing one of them. Thus, Lasso may be more useful with a large set of features where some redundant ones could be eliminated or pruned and which would lead to better prediction. Ridge may be more useful when there is a large number of highly correlated features where shrinking them together without eliminating any would lead to better prediction.

2.5 Prediction Using Hold-Out Test Set

After validation, the best prediction model can be chosen to perform prediction using the test data set or the hold-out data set. This is a final check on the model to see that if it performs just as well in the test set, then it can be rolled out to perform prediction with generalized data.

We perform the optional step of combining the training data set and the validation data set for re-training the model parameters, but using the fine-tuned hyperparameters from the validation phase. The re-trained model is then

applied to the prediction of targets from the test data set using the test data features.

Code line [29] shows how the step is done.

```
[29]: Ridge=linear_model.Ridge(alpha=alpha_Ridge,)
      Ridge.fit(X_trainval, y_trainval)
      y_pred_Ridge_test = RI + np.dot(X_test,RC.T)
      r2_score_Ridge_test = r2_score(y_test, y_pred_Ridge_test)
      rmse_y_pred_Ridge_test = (np.sqrt(mean_squared_error(y_test, y_pred_Ridge_test)))
      print("r2_score_Ridge_test:",r2_score_Ridge_test)
      print("Pred_RMSE_Ridge_test:", rmse_y_pred_Ridge_test)

      Lasso=linear_model.Lasso(alpha=alpha_Lasso,)
      Lasso.fit(X_trainval, y_trainval)
      y_pred_Lasso_test = LI + np.dot(X_test,LC.T)
      r2_score_Lasso_test = r2_score(y_test, y_pred_Lasso_test)
      rmse_y_pred_Lasso_test = (np.sqrt(mean_squared_error(y_test, y_pred_Lasso_test)))
      print("r2_score_Lasso_test:",r2_score_Lasso_test)
      print("Pred_RMSE_Lasso_test:", rmse_y_pred_Lasso_test)

      ELNet=linear_model.ElasticNet(alpha=alpha_EN, l1_ratio=l1_ratio_EN,)
      ELNet.fit(X_trainval, y_trainval)
      y_pred_ELNet_test = EI + np.dot(X_test,EC.T)
      r2_score_ELNet_test = r2_score(y_test, y_pred_ELNet_test)
      rmse_y_pred_ELNet_test = (np.sqrt(mean_squared_error(y_test, y_pred_ELNet_test)))
      print("r2_score_ELNet_test:",r2_score_ELNet_test)
      print("Pred_RMSE_ELNet_test:", rmse_y_pred_ELNet_test)

      r2_score_Ridge_test: 0.6741670178819545
      Pred_RMSE_Ridge_test: 779.7022050419647
      r2_score_Lasso_test: 0.6741543806451812
      Pred_RMSE_Lasso_test: 779.7173250365355
      r2_score_ELNet_test: 0.6741681342698304
      Pred_RMSE_ELNet_test: 779.7008693103447
```

In [29], we find the test data prediction scores for all 3 models since they perform with marginal differences in the validation phase. As seen in the output, the Elastic Net regression model is now marginally better than that of the Ridge regression. The performance at R^2 of over 67% is close to the validation R^2 performance of just over 69%.

2.6 Cross-Validation

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. K-fold cross validation refers to the number of groups that a given data sample less the hold-out test sample is split into. This method allows validation to be conducted within the training data set itself and thus avoids use of another separate set of data for validation.

k approximately equal subsets of the data sample after leaving out the hold-out set are drawn randomly. Pick a subset as the validation sample. Pick the rest of subsets as training sample. Fit model on training sample and evaluate prediction R^2 on the validation sample. Pick the next subset as the validation sample, and the rest as training sample, and so on. There will be k number of such fittings and k number of validation R^2 scores using this k-fold cross-validation approach. The mean and variance of the k number of R^2 scores can be computed.

A different set of hyperparameter value(s) is used for each cross-validation mean R^2 score. The optimal hyperparameter value(s) is/are that which yield(s) the maximum mean R^2 score in the cross-validation. This optimal hyperparameter value(s) is/are then used to check the prediction performance of the test data set. The cross-validation procedure essentially replaces 0.6N, 0.2N, 0.2N slicing of training, validation, and test data sets with 0.8N, 0.2N simpler slicing into training and test data sets. If k=5, then the training data set is divided into five times of 0.64N training and 0.16N validation.

Repeated k-fold cross-validation occurs when the resampling is done again by randomly splitting the sample into the hold-out and remainder and randomly selecting the k subsets again. Stratified cross-validation occurs when it is ensured that each subset of the data sample contains the same proportion of observations with a given categorical value, e.g., stratification by gender means each subset should have the same proportion of gender mix. Generally, increasing k and/or increasing the number of repetitions will reduce the bias (mean of the scores departure from true mean) but increase the variance or noise of the scores.

There is some similarity of the concept of cross-validation in ML with resampling schemes in classical statistics. The Jackknife is a method when sequentially one observation is left out and the required statistic is computed based on the remaining data. Sample of N requires N number of such computations. Mean and variance of the various computed statistics provide idea of closeness to true statistic and its variance. This idea is similar to cross-validation. Bootstrapping is a statistical procedure that resamples a single dataset to create many simulated samples. This process allows you to calculate standard errors, construct confidence intervals, and perform hypothesis testing for numerous types of the sample statistics. This is similar in concept to repeated cross-validation. While Jackknife and Bootstrapping are resampling procedures to compute

sample statistics, cross-validation and repeated cross-validation are procedures to compute mean validation scores.

The discussion on training, validation, and testing can be depicted in Figure 2.1.

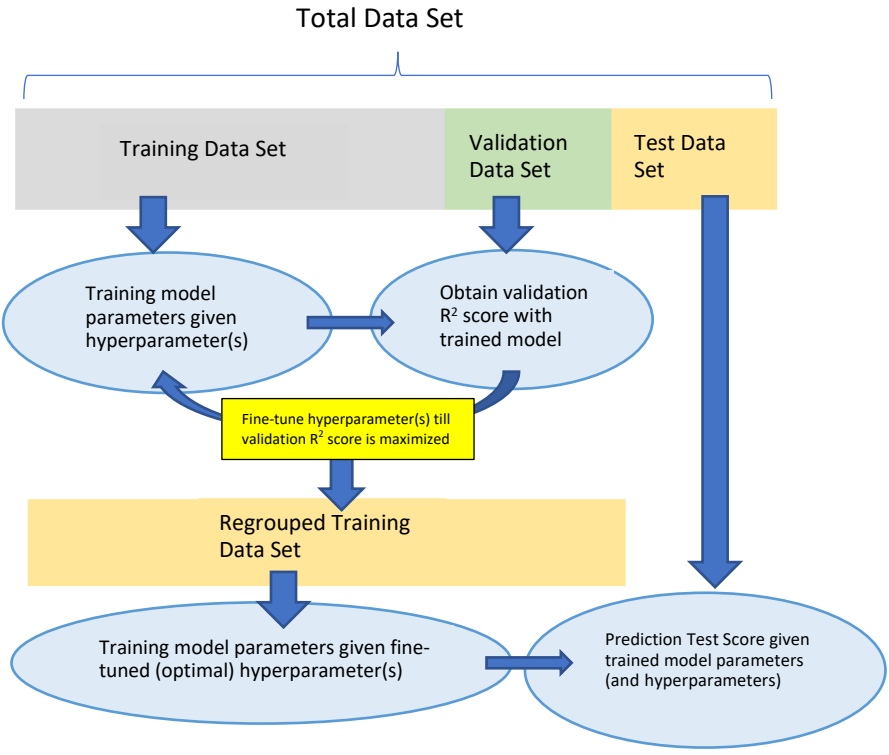


Figure 2.1

The k-fold cross-validation is a common approach or strategy to improve validation effectiveness and thus comes up with an optimal model (with optimized model parameters as well as hyperparameters) for checking prediction using the test data set that has been held-up. This is done particularly with smaller sample size (smaller number of sample points) in the total data set. It is, however, usually not applicable when the data are in time series form as time ordering cannot be randomly re-arranged. In what follows, we also show diagrammatically how hyperparameters can be fine-tuned or optimally selected using this k-fold cross-validation approach.

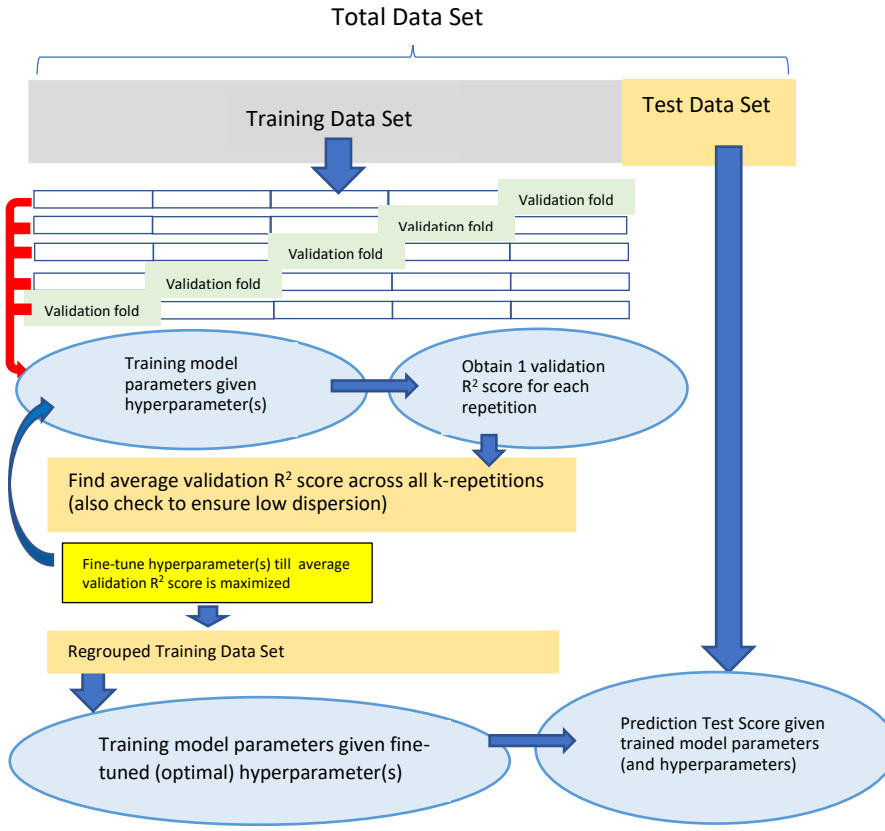


Figure 2.2

In Figure 2.2, it is shown that the original training data set and validation data set are combined into a larger training data set. This set is then divided into k-folds. k is typically chosen as 5 or 10 although in theory it can also be a hyperparameter. Suppose we choose k=5. Thus, there are 5 validations. Note that each data point appears only once in the validation data set of all the k repetitions. There are k number of validation R^2 scores. These are averaged to get the mean cross-validation scores and a standard deviation or dispersion of the scores is also obtained. The hyperparameters are fine-tuned till the average cross-validation score is maximized. This selected optimal hyperparameter value(s) is then used to train the model parameters in the regrouped training data set. This is then applied to the test data set to check for hold-out data or

out-of-sample prediction accuracy as the final step to machine learning in this case.

Code line [30] shows that the data set could be reshuffled anew with a fresh X_train and y_train training set of 49,887 sample points. Code line [33] provides the cross-validation results for the base case of linear regression without constraint.

```
[33]: ### Now k-fold cross validation is to be performed on X_train y_train
      ###  reshuflled dataset, leaving test set intact
      ### Details of other scoring methods and metrics can be found in
      ###  https://scikit-learn.org/stable/modules/model_evaluation.html

      from sklearn.model_selection import cross_val_score

      scoresLinreg = cross_val_score(estimator = Linreg, X = X_trainval, \
                                     y = y_trainval, cv = 5)
      ### This combined training set X_train, y_train is split into k=5
      ###  (cv=5) folds for each of k=1,2,...,5 repetitions
      print(scoresLinreg)
      ### Score is R2 measure, there are 5 scores since k=cv=5,
      ###  one for each repetition

      print("%0.8f mean R2 with a standard deviation of %0.8f" \
            % (scoresLinreg.mean(), scoresLinreg.std()))

[0.66375598 0.67706077 0.67908507 0.67785626 0.69054729]
0.67766107 mean R2 with a standard deviation of 0.00850685
```

The computations according to Figure 2.2 (in this case without tuning any hyperparameter) is performed using cross_val_score module in sklearn. k = 5 is used here (see 'cv = 5'). The mean validation R^2 score is 0.67766107 with a standard deviation of 0.008507 in the scores. This mean score is slightly lower than 0.69452567 in the case without cross validation, though this result could be more robust with random draws in the training versus test samples.

Code lines [35], [37], [39] show the cross-validation scoring results based on k=5 for Ridge, Lasso, and Elastic Net regressions. The optimal hyperparameter is found using GridSearchCV in sklearn.model_selection.

Summary of the results are shown as follows.

	Mean Validation R^2 Score	Std. Dev. of Validation R^2 Scores
Ridge (α = 0.53)	0.67766112	0.008509
Lasso (α = 0.0)	0.67766107	0.008507
Elastic Net (α = 0.001, l1_ratio = 0.99)	0.67766111	0.008508

As seen in the cross-validation results using R^2 as the metric or measurement of the accuracy of prediction performance, Ridge regression performs marginally better than regression without constraint and also better than Lasso and Elastic Net, in terms of mean R^2.

Cross-validation for Ridge Regression

[35]:
```
### Now k-fold cross validation is to be performed on X_train y_train dataset
###  using Ridge regression

from sklearn import linear_model
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

Ridge_range = [i/100.0 for i in range(0,100)]
parameters = {'alpha': Ridge_range}

model_Ridge = Ridge()
Ridge_grid_search = GridSearchCV(estimator=model_Ridge,
                         param_grid=parameters,
                         cv=5)

Ridge_grid_search.fit(X_trainval, y_trainval)

# Print the best hyperparameters and the corresponding mean cross-validated score
print("Best Hyperparameters: ", Ridge_grid_search.best_params_)
print("Best Score: ", Ridge_grid_search.best_score_)
print("Std Err of Scores at Best Hyperparameter: ", \
      Ridge_grid_search.cv_results_['std_test_score'][Ridge_grid_search.best_index_])
```

```
Best Hyperparameters:  {'alpha': 0.53}
Best Score:  0.6776611238244105
Std Err of Scores at Best Hyperparameter:  0.008508623139690662
```

Cross-validation for Lasso Regression

[37]:
```
### Now k-fold cross validation is to be performed on X_train y_train dataset
###  using Lasso regression

from sklearn import linear_model
from sklearn.linear_model import Lasso

Lasso_range = [i/100.0 for i in range(0,30)]
parameters = {'alpha': Lasso_range}

model_Lasso = Lasso()
Lasso_grid_search = GridSearchCV(estimator=model_Lasso,
                         param_grid=parameters,
                         cv=5)

Lasso_grid_search.fit(X_trainval, y_trainval)
```

```
# Print the best hyperparameters and the corresponding mean cross-validated score
print("Best Hyperparameters: ", Lasso_grid_search.best_params_)
print("Best Score: ", Lasso_grid_search.best_score_)
print("Std Err of Scores at Best Hyperparameter: ", \
      Lasso_grid_search.cv_results_['std_test_score'][Lasso_grid_search.best_index_])
```

```
Best Hyperparameters:  {'alpha': 0.0}
Best Score:  0.6776610737723814
Std Err of Scores at Best Hyperparameter:  0.00850685083401788
```

Cross-validation for Elastic Net Regression

[39]:
```
### Now k-fold cross validation is to be performed on X_train y_train dataset
###  using Elastic Net regression

from sklearn import linear_model
from sklearn.linear_model import ElasticNet

alpha_range = [i/1000.0 for i in range(0,100)]
l1_ratio_range = [i/100.0 for i in range(0,100)]
param_grid = {'alpha': alpha_range, 'l1_ratio': l1_ratio_range}

model_ELNet = ElasticNet()
ELNet_grid_search = GridSearchCV(estimator=model_ELNet,
                         param_grid=param_grid,\
                         cv=5)

ELNet_grid_search.fit(X_trainval, y_trainval)

# Print the best hyperparameters and the corresponding mean cross-validated score
print("Best Hyperparameters: ", ELNet_grid_search.best_params_)
print("Best Score: ", ELNet_grid_search.best_score_)
print("Std Err of Scores at Best Hyperparameter: ", \
      ELNet_grid_search.cv_results_['std_test_score'][ELNet_grid_search.best_index_])
```

```
Best Hyperparameters:  {'alpha': 0.001, 'l1_ratio': 0.99}
Best Score:  0.6776611126209133
Std Err of Scores at Best Hyperparameter:  0.008508289316292484
```

Using the process as in Figure 2.2, the eventual use of the best method, whether linear regression or the regularized regression, together with the optimally selected hyperparameter(s), may then be applied to the regrouped training data and the computed model is then applied in the final step to the test data for prediction accuracy assessment. This is shown in code lines [41], [42], and [43].

[41]:
```
### Computing for Ridge
Ridge2=linear_model.Ridge(alpha=0.53)
Ridge2.fit(X_trainval, y_trainval)
y_pred_Ridge2_test =  Ridge2.intercept_ + np.dot(X_test,Ridge2.coef_.T)
r2_score_Ridge2_test = r2_score(y_test, y_pred_Ridge2_test)
rmse_y_pred_Ridge2_test = (np.sqrt(mean_squared_error(y_test, y_pred_Ridge2_test)))
print("r2_score_Ridge2_test:",r2_score_Ridge2_test)
print("Pred_RMSE_Ridge2_test:", rmse_y_pred_Ridge2_test)
```

```
r2_score_Ridge2_test: 0.6741727289444343
Pred_RMSE_Ridge2_test: 779.6953718670477
```

[42]:
```
### Computing for Lasso
Lasso2=linear_model.Lasso(alpha=0.0)
Lasso2.fit(X_trainval, y_trainval)
y_pred_Lasso2_test =  Lasso2.intercept_ + np.dot(X_test,Lasso2.coef_.T)
r2_score_Lasso2_test = r2_score(y_test, y_pred_Lasso2_test)
rmse_y_pred_Lasso2_test = (np.sqrt(mean_squared_error(y_test, y_pred_Lasso2_test)))
print("r2_score_Lasso2_test:",r2_score_Lasso2_test)
print("Pred_RMSE_Lasso2_test:", rmse_y_pred_Lasso2_test)
```

```
r2_score_Lasso2_test: 0.6741702504690269
Pred RMSE Lasso2 test: 779.6983373219088
```

[43]:
```
### Computing for Elastic Net
ELNet2=linear_model.ElasticNet(alpha=0.001,l1_ratio=0.99,)
ELNet2.fit(X_trainval, y_trainval)
y_pred_ELNet2_test =  ELNet2.intercept_ + np.dot(X_test,ELNet2.coef_.T)
r2_score_ELNet2_test = r2_score(y_test, y_pred_ELNet2_test)
rmse_y_pred_ELNet2_test = (np.sqrt(mean_squared_error(y_test, y_pred_ELNet2_test)))
print("r2_score_ELNet2_test:",r2_score_ELNet2_test)
print("Pred_RMSE_ELNet2_test:", rmse_y_pred_ELNet2_test)
```

```
r2_score_ELNet2_test: 0.6741727740241151
Pred RMSE ELNet2 test: 779.6953179298503
```

2.7 Concluding Thoughts

There are clearly advantages in the use of regularized regressions as a comparison to the linear regression model without constraint when the model may have too many features and can be overfitted using the training data. There are many studies involving proprietary data sets showing advantages of or better performances of regularizations – see Jorge Chan-Lau (2017), and Xin and Khalid (2018).

When the prediction problem does not have too many features, then using regularizations may not significantly improve prediction performance. Often, data features that may have large differences in their values require scaling of the features to improve the performance of the method. A min_max_scaler = preprocessing.MinMaxScaler() can be used where each feature is scaled to (X - X.min) / (X.max - X.min), i.e., within (0,1). Or else, a normalizing method such as sklearn.preprocessing.StandardScaler (*, copy=True, with_mean = True, with_std=True) can be used. The features are transformed by subtracting the sample mean from each and dividing by the corresponding standard deviation. Scaling is not done in the example in this chapter as the features are mostly binary cases of '1's or '0's and the others do not have extreme values.

In this chapter we show how a validation data set is used to fine-tune hyperparameters to obtain maximum performance in the prediction of validation targets based on training the parameters in the training data set. We

also show how when the data sample size is small, a cross-validation method can be used to accompany regularized regressions. The advantage of cross-validation is that its results are likely to be more robust, e.g., in selecting optimal hyperparameter(s) by considering average of the validation R^2 scores. However, cross-validation may take up more computing time with the number of repetitions, especially with larger data sets.

The data example in this chapter is to show how regularization can be done. By today's data standards, for predicting real estate values professionally, a lot more features can and should be added such as: recently transacted values of similar neighboring plots or blocks or units, distances/connections to transport nodes such as highways, train/bus stations, distances/connections to markets/shops/hospitals/entertainment centers, distances to childcare facilities, distances to seaside or vista points, neighborhood crime rates/theft rates, parking facilities and estate road and amenities conditions, electricity/gas supply conditions, weather conditions including flooding, hurricane situations, and so on. These additional features are likely to produce better regularized regression predictions.

2.8 Other References

Jorge A. Chan-Lau, (2017), "Lasso Regressions and Forecasting Models in Applied Stress Testing", IMF Working Paper, Institute for Capacity and Development.

Seng Jia Xin, and Kamil Khalid, (2018), "Modelling House Price Using Ridge Regression and Lasso Regression", International Journal of Engineering and Technology, 7, 498-501.

https://pythonprogramming.net/features-labels-machine-learning-tutorial/#:~:text=How%20does%20the%20actual%20machine,attempting%20to%20predict%20or%20forecast.

https://www.louisaslett.com/StatML/notes/error-estimation-and-model-selection.html

https://vitalflux.com/ridge-regression-concepts-python-example/

https://towardsdatascience.com/k-fold-cross-validation-explained-in-plain-english-659e33c0bc0

https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a