2 REGRESSION AND REGULARIZATION

In this chapter, we discuss predicting a target variable using (multiple) linear regression on a number of features (or explanatory variables) possibly related to the target. We also explain the use of different data sets for training, validation, and testing in conjunction with model hyperparameters. Model parameters are parameters (carrying numerical values) that are part of the theoretical or distributional model. They are part of what defines the actual or true model. These are important in classical econometrics or statistics where the true model or distribution under the null hypothesis often provides direct implications on the estimation and testing. In machine learning, which may be analogous to a corn harvesting machine on a corn field picking up corn and discarding chaff, the nuts and bolts and width of the roller in the machine are the hyperparameters. The latter do not determine the true corn proportion amongst the corn and chaff but are important for fine-tuning to be able to yield the highest pickup rate for corn and low pickup of chaff. In this chapter, a useful method of cross-validation in machine learning is also discussed.

Suppose there are n number of observed target variables and n sets of associated features. We can set aside approximately 80% of data or 0.8n for training and the remainder 20% or 0.2n for testing. Or select sample sizes that are integers closest to 0.8n, 0.2n. Training is basically to train and select optimal values of the regression coefficients in order to perform the testing.

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

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

where ε_k is a residual variable that is unobserved. Unlike 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. 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, $\sum_{k=1}^{0.8n}(Y_k - \hat{Y}_k)^2$ is minimized, where fitted $\hat{Y}_k = \hat{b}_0 + \hat{b}_1 X_{1k} + \hat{b}_2 X_{2k} + \cdots + \hat{b}_p X_{pk}$. R^2-score for the training data set is 1 – $\sum_{k=1}^{0.8n}(Y_k - \hat{Y}_k)^2 / \sum_{k=1}^{0.8n}(Y_k - \overline{Y}_k)^2$. 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 the test data set of 0.2n number of the remaining Y_k's, using the associated test 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 test 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 test data is different so that $\sum_{k=1}^{0.2n}(Y_k - Y_k^*)^2$ is very large. The R^2 score in the test 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 in a test data set, assuming both data sets come from the same data generating model. The use of training and test data sets is similar to the idea of in-sample and out-of-sample fits.

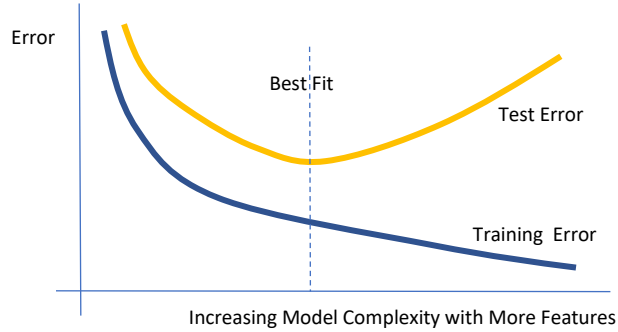
It should be noted that where $\{Y_k, X_{1k}, X_{2k}, X_{3k}, \ldots, X_{pk}\}_{k=1,2,\ldots n}$ is not a time series, but a cross-section, then splitting the set into training and testing subsets can be random. However, if it is a time series, then typically the training set data would precede those 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 test set 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 prediction using the test data set.

The overfitting could occur with some outlier features that attract coefficients with high fitted values. Overfitting could also occur when using too many features often brings about some features that are highly multi-collinear, i.e., having high correlations. The high multi-collinearity could produce very high fitted coefficients in some features while low ones in others, leading to poor prediction out-of-sample.

Underfitting occurs when both the R^2's in the training data set fitting and in the test data set prediction are low. There is possibly an ideal balance between overfitting and underfitting as seen in the graph below.



To reduce overfitting and enable good prediction, i.e., high R^2's in the training data set fitting and in the test data set prediction, some constraints are added to the regression.

Prediction error or deviation of actual test data value from predicted value could typically be produced by too large values of some fitted coefficients that were too eager to overfit the training data set or due to multi-collinearity. A penalty function is added to reduce the tendencies to yield large-fitted coefficients – this technique is called regularization. There are two common regularized linear regression methods – Ridge regression and Lasso (Least absolute shrinkage and selection operator) regression. Regularization could shrink some fitted coefficient 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 incorrect outsized outliers and also high multi-collinearity.

For Ridge regression using the module sklearn.linear_model.Ridge, see documentation in https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html. One can also check the [source] for the explicit

codes. This module is the Ridge regression model that minimizes the objective function

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

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. When we use the computed value of linear_model. Ridge ( ), the default parameters within ( ) includes 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 the [source] for the explicit codes. This module is the Lasso regression model that minimizes the objective function

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

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 computed value of linear_model.Lasso ( ), the default parameters within ( ) includes setting alpha > 0, e.g., $\alpha = 0.1$. We can of course try different values for the alpha hyperparameter to attain low training error.

2.2 Worked Example – Data

We explain the method behind one of the popular problems posted on public websites – that of predicting California house prices. 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.

This publicly accessible dataset comprises only 8 features (characteristics) of houses in the various areas of the state of California in a 1990 US Census survey. 20,640 samples (sample points) are available in the data set California_housing.frame.csv, each of which is a residential block in

California. The target variable is MedHouseVal -- median house value in a block, in units of US$100,000. The features are:

(1) MedInc -- median of household incomes in block, in units of $10,000,
(2) HouseAge – median age of house in block in years,
(3) AveRooms -- average number of rooms per household in block,
(4) AveBedrms -- average number of bedrooms per household in block,
(5) Population -- total number of occupants in block,
(6) AveOccup -- average number of occupants per household in block,
(7) Latitude, and
(8) Longitude.

The smaller or more negative longitude (124º West is represented by –124º) implies locations closer to the sea. The cross-sectional data consist of 20,640 blocks of residential houses. The 8 features can be used as explanatory variables for the target price variable. Some public resources on this topic can be accessed at URLs such as

https://inria.github.io/scikit-learn-mooc/python_scripts/datasets_california_housing.html
https://www.kaggle.com/datasets/camnugent/california-housing-prices
https://www.classes.cs.uchicago.edu/archive/2021/fall/12100-1/pa/pa5/dataset-houseprice.html
https://developers.google.com/machine-learning/crash-course/california-housing-data-description

As the 'MedHouseVal' price data might have been right-censored at 5 (representing $500,000), any number above 5 is entered as 5. The censored data may not carry the correct information for purpose of analyses since a $10 million median house value is recorded as $500,000 and may distort the fitting and testing results. We remove such sample points as they do not carry the correct price information. We also remove incorrect data of 'AveOccup' > 10. This leaves 19,615 sample points. In what follows, we use the Regression method and regularization to predict the California house price given the relevant features of that house within a block. The Jupyter codes for exploring the data are shown below.

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

In [2]:
```
### We can import the data set in another way via a downloaded csv file in the same Dir as this Jupyter kernel (program file)
### The dataframe name is now df
df = pd.read_csv('california_housing.frame.csv')

### There is an aberration in the large spikes of up to 1200 occupants per household in the original data set.
### These could be incorrectly notated. We delete 37 such records with AveOccup > 10
df = df[~(df['AveOccup'] >= 10.0)]

df=df.iloc[:,1:10]
```

In [3]:
```
df.head()
### This returns the first 5 rows of the dataframe df
```

Out[3]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

In [4]:
```
np.shape(df)
### This returns the dimension of the tabular dataframe, i.e. 506 rows of data without variable names, 15 columns excepting index
```

Out[4]:
```
(20603, 9)
```

In [5]:
```
# Let's summarize the data to see the distribution of data
print(df.describe())
### The behaviour of describe() is different with series of strings. giving count of values, top and freq of occurrence
```

```
              MedInc      HouseAge      AveRooms     AveBedrms    Population  \
count  20603.000000  20603.000000  20603.000000  20603.000000  20603.000000
mean       3.870548     28.637092      5.428533      1.096540   1423.339368
std        1.897012     12.580805      2.471073      0.473866   1128.241516
min        0.499900      1.000000      0.846154      0.333333      3.000000
25%        2.564350     18.000000      4.440829      1.006135    787.000000
50%        3.534900     29.000000      5.229682      1.048749   1166.000000
75%        4.743850     37.000000      6.052108      1.099418   1724.000000
max       15.000100     52.000000    141.909091     34.066667  35682.000000

           AveOccup      Latitude     Longitude   MedHouseVal
count  20603.000000  20603.000000  20603.000000  20603.000000
mean       2.918517     35.631012   -119.569189      2.069010
std        0.765234      2.135875      2.003460      1.153654
min        0.692308     32.540000   -124.350000      0.149990
25%        2.428935     33.930000   -121.800000      1.196000
50%        2.817259     34.260000   -118.490000      1.798000
75%        3.279038     37.710000   -118.010000      2.649000
max        9.954545     41.950000   -114.310000      5.000010
```

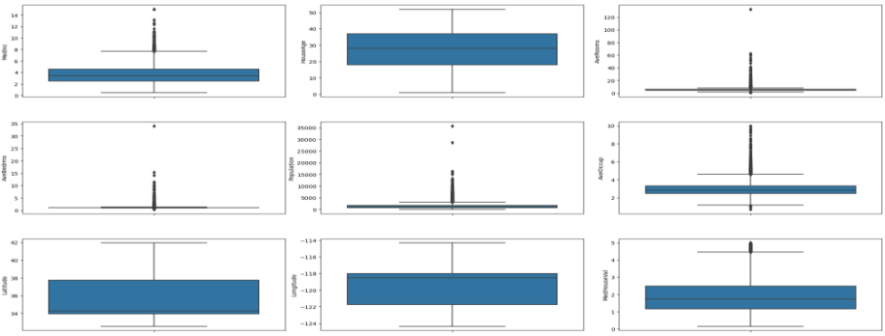
In [8]:
```
### MedHouseVal is censored at 5, i.e. numbers above 5 are reported as 5. This number is therefore inaccurate. Remove these.
df = df[~(df['MedHouseVal'] >= 5.0)]
print(np.shape(df))
```

```
(19615, 9)
```

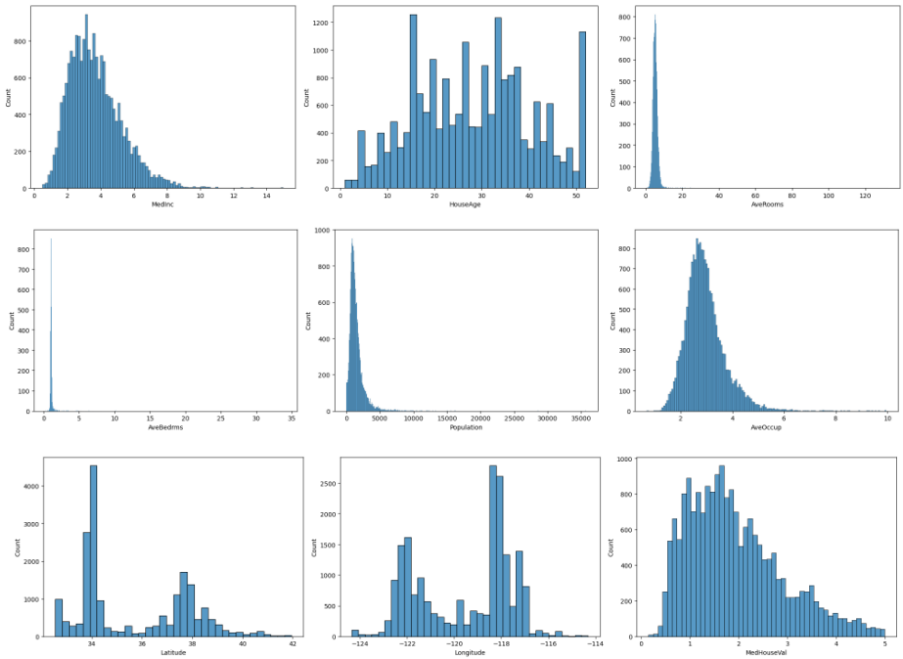
From code lines [9] to [11], we explore the empirical distributions of each feature. Such preliminary explorations provide some ideas of which features may be more important in the prediction.

In [9]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats

fig, axs = plt.subplots(ncols=3, nrows=3, figsize=(20, 10))
index = 0
axs = axs.flatten()
for k,v in df.items():
    sns.boxplot(y=k, data=df, ax=axs[index])
    index += 1
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=5.0)
### In the boxplot, centerline is mean, upper and lower box edges are the 75th and 25th percentiles.
### Whiskers connect to "max" [75% + 1.5* interquartile range IQR] and "min" [25% - 1.5* IQR] (shorter horizontal lines)
### Outliers are shown as dots outside the whiskers.
### See also https://seaborn.pydata.org/generated/seaborn.boxplot.html
```



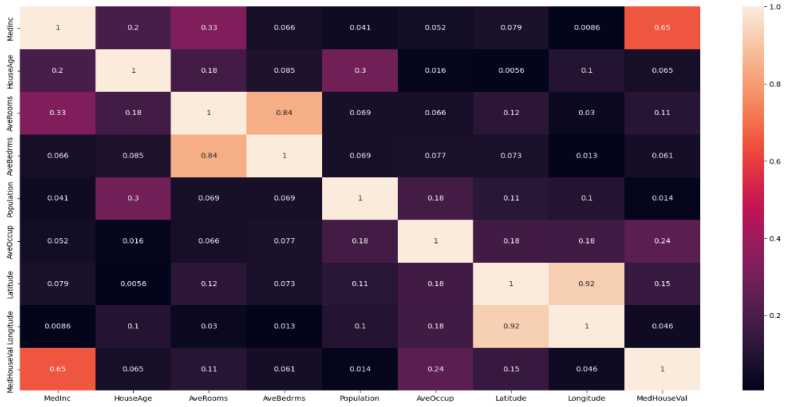
In [10]:
```
fig, axs = plt.subplots(ncols=3, nrows=3, figsize=(20, 15))
index = 0
axs = axs.flatten()
for k,v in df.items():
    sns.histplot(v, ax=axs[index])
    index += 1
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=5.0)
### uses seaborn
```



Code line [13] shows the correlation heat map amongst the features, including the target variable. It is seen that MedInc has reasonably high correlation with MedHouseVal, followed by AveOccup, Latitude and AveRooms. The other features have low pairwise correlations with MedHouseVal, below 0.1.

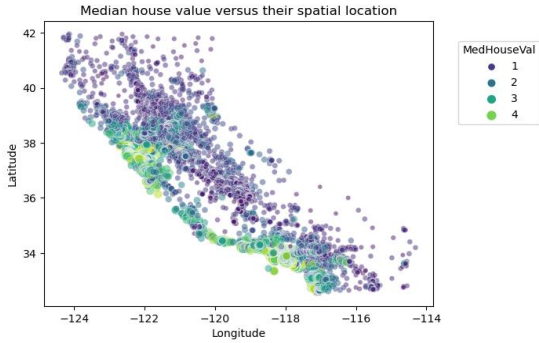
```
In [13]: ### pairwise correlation on the features including medv
         plt.figure(figsize=(20, 10))
         sns.heatmap(df.corr().abs(),  annot=True)
         ### In heat map, clearly tax and rad are highly correlated
         ### For more options - see https://seaborn.pydata.org/generated/seaborn.heatmap.html
```

Out[13]: <AxesSubplot:>



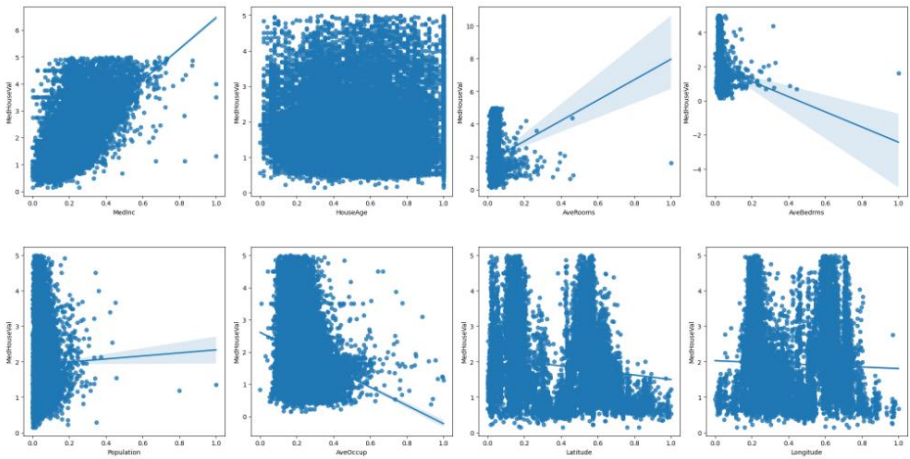
The 3-dimensional relationship of Latitude- Longitude and associated MedHouseVal (shown in color dots) is shown using seaborn graphics. Clearly coastal blocks are generally more valuable.

```
In [14]: import seaborn as sns
         sns.scatterplot(data=df, x="Longitude", y="Latitude",
                         size="MedHouseVal", hue="MedHouseVal",  ###size is grouping variable, hue is order of proecessing
                         palette="viridis", alpha=0.5)
         plt.legend(title="MedHouseVal", bbox_to_anchor=(1.05, 0.95),
                    loc="upper left")
         _ = plt.title("Median house value versus their spatial location")
         ### key shows MedHouseVal in ranges (1) 0-1, (2) 1-2, (3) 2-3, (4), > 3
```



Next we scale the features to (0,1) via sklearn preprocessing app. The resulting pairwise graphs of scaled feature with the target variable are shown below. Only the MedInc graph shows a more apparent correlation. The overall preliminary visual aids appear to indicate weak regression or predictive results.

```
In [17]: from sklearn import preprocessing
         # Let's scale the columns before plotting them against MedHouseVal
         min_max_scaler = preprocessing.MinMaxScaler()
         ### X is now transformed via min_max_scaler as (X - X.min) / (X.max - X.min)
         column_sels = ['MedInc','HouseAge', 'AveRooms', 'AveBedrms','Population', 'AveOccup', 'Latitude','Longitude']
         ### Note varnames must be in exact small or cap letters
         ### Features are included in order of their pairwise correlation with MedHouseVal
         X = df.loc[:,column_sels]
         y = df['MedHouseVal']
         X = pd.DataFrame(data=min_max_scaler.fit_transform(X), columns=column_sels)
         fig, axs = plt.subplots(ncols=4, nrows=2, figsize=(20, 10))
         index = 0
         axs = axs.flatten()
         for i, k in enumerate(column_sels):
             sns.regplot(y=y, x=X[k], ax=axs[i])
         plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=5.0)
         ### Note: sns.pairplot(dataset); plt.show() would produce plots of all pairs
```



2.3 Linear Regression Prediction

A linear regression model will be used to predict the housing prices in subsection 2.2. For this, a standard procedure would be to split the 19,615 data points (each point referring to a set of $\{Y_k, X_{1k}, X_{2k}, X_{3k}, \ldots, X_{pk}\}$values for a given k) into a training data set, a validation data set, and a test data set. This could be in proportions 60%, 20%, and 20% respectively. Regressions are done on the training data set using several models, each of which may involve a different set of features or explanatory variables. The best fitted model is usually chosen in terms of the highest R^2 or coefficient of

determination. In linear regression, R^2 = 1 – residual sum of squared errors/total sum of squared errors.

The fitted models are then used to predict the y values in the validation data set. This allows optimizing or fine-tuning of hyperparameters in the model if any to find the model with highest R^2 or best fit. Overfitted models in the sense of poor R^2 in the validation data set are to be removed, and the best model in terms of highest R^2 would be used in testing. Before touching the test data set or the hold-out data set, the original training and validation data sets could be combined to do training once more for the chosen model. The latest fitted model is then used to check out prediction score or R^2 in the test data set. If the latter test score is reasonably high (close to the training R^2), then the model is ready for use to predict the next unseen set of new features or generalized data.

We shall assume that the model with all the 8 features is the optimal one after validation, and that the model is going to be estimated or fitted next using the 60%+20% data sets (step before applying testing data). In Scikit-learn (or simply sklearn) module, there is a useful train_test_split function that splits the total data set into just a training set and a test set. We use this function to split into the 80% training and 20% test accordingly as explained above. See code line [18]. There is a random selection of the 80% versus 20%. However, if it is a time series, then random selection is not suitable.

```
In [18]: # Splitting the dataset into the Training set and Test set
         from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 50)
         ### test_size here is 20%; random state number is a seed number -- diff seed produces diff random draws
         ### note 0.2*19615 = 3923 (or closest to an integer). 0.8*19615 = 15,692.
         ### X_train, y_train each has 15,692 rows.
         ### X_test, y_test each has 3923 rows
         ### Latter can be checked using print(len(y_test))
         print(len(X_train),len(y_train),len(X_test),len(y_test))

         15692 15692 3923 3923
```

In code line [19], y_train is regressed linearly on X_train and the estimated intercept (train) and slopes (train) coefficients are seen in the outputs. Training set R2_score (train) is 0.62035, i.e. $1 - \sum_{k=1}^{0.8n}(Y_k - \hat{Y}_k)^2 / \sum_{k=1}^{0.8n}(Y_k - \overline{Y}_k)^2$. $\hat{Y}_k$ is fitted Y_k using the estimated intercept (train) and slopes (train) coefficients multiplied by the training set $\{X_{1k}, X_{2k}, X_{3k}, \ldots, X_{pk}\}$values.

```
In [19]: ### see module documentation in https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
         ### LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=True)
         ### copy_X=True -- X will be copied; else, it may be overwritten.
         ### if fit_intercept = False, intercept will be set to 0.0
         ### n_jobs=None (same as  = 1), i.e. no use of additional parallel processors
         ### normalize=False: ignored when fit_intercept is set to False.
         ### If True, the regressors X will be normalized before regression by subtracting the mean and dividing by the l2-norm.
         ### If you wish to standardize, one could use StandardScaler before calling fit on an estimator with normalize=False.

         from sklearn.linear_model import LinearRegression
         Linreg = LinearRegression()
         Linreg.fit(X_train, y_train) ###training the algorithm, note regression is done without normalizing the X, y
         ### 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)
         ### Fitting y using the x_train data
         r2_score_Linreg_train = r2_score(y_train, y_pred_Linreg_train)

         print('R2_score (train): ', r2_score_Linreg_train)
         ### R2_score (train) is the R-square in the linear regression involving only the training data set

Intercept (train): 4.077726011109183
Slopes (train): [  5.91059822   0.46819653 -10.58017655  15.28770265   1.30115905
  -2.65806525  -3.69542188  -3.92228473]
R2_score (train):  0.6203540883823073
```

The training criterion of root mean square error is seen as minimized at 0.59749 in code line [20]. Note that RMSE is not R^2. If RSS = Residual sum of squared errors, then RMSE = $\sqrt{}$(RSS/N) where N is the sample size.

```
In [20]: 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("RMSE: ", rmse_Linreg_train)

RMSE:  0.5974983971900051
```

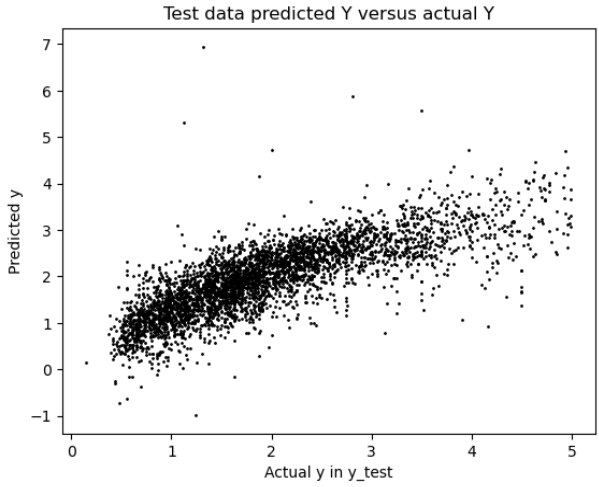
In code line [22], Y_k^* is predicted value of Y_k in the test data set using $Y_k^* = \hat{b}_0 + \hat{b}_1 X_{1k} + \hat{b}_2 X_{2k} + \cdots + \hat{b}_p X_{pk}$ where $\hat{b}_0$ is intercept (train) and $\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_p$ are the slopes coefficients from the fitting in the training set. X_{jk}'s (for j =1,2,…,p) are from the test data set. The test score is $R^2 = 1 - \sum_{k=1}^{0.2n}(Y_k - Y_k^*)^2 / \sum_{k=1}^{0.2n}(Y_k - \overline{Y}_k)^2$ where Y_k is y_test and Y_k^* is y_pred. Test score R^2 is seen as 0.62137 which is slightly higher than the training R^2 shown in [19]. Another metric in measuring prediction performance besides R^2 is the root mean square error (RMSE) which is $\sqrt{\sum_{k=1}^{0.2n}(Y_k - Y_k^*)^2/(0.2n)}$ where Y_k is in the test data set. Here RMSE is 0.60128. The total sum of squares of Y_k in the test data set is $\sum_{k=1}^{0.2n}(Y_k - \overline{Y}_k)^2$or tss(y_test). Note that the prediction test score R^2 can also be computed using 1 = 0.2n * $RMSE^2$/tss(y_test). These results are seen in code line [22].

```
In [22]: ### Predicting R2 Score using Test Set but Intercept(train) and Slopes(train) from Training results
         y_pred = Linreg.intercept_ + np.dot(X_test,Linreg.coef_.T)
         rmse_y_pred = (np.sqrt(mean_squared_error(y_test, y_pred)))
         def tss(y_test):
             return ((y_test - np.mean(y_test))**2).sum()
         R2_pred=1-(3923*(rmse_y_pred)**2)/tss(y_test)
         print("Pred_RMSE:",rmse_y_pred)
         print("R2_pred:",R2_pred)

         Pred_RMSE: 0.6012841104200348
         R2_pred: 0.6213764009165903
```

The graph of the test data predicted y values versus the actual y (y_test values) is shown in the scatterplot of [24]. The prediction is seen to be reasonably good.

```
In [24]: ### A scatterplot may be clearer - here matplotlib
         plt.scatter(y_test, y_pred,s=1,color='black')
         plt.title("Test data predicted Y versus actual Y")
         plt.xlabel("Actual y in y_test")
         plt.ylabel("Predicted y")
         plt.show()
```



Next, we apply Ridge regression and Lasso regression. These are shown in code lines [25] to [28]. In both regularized regressions, the regularization penalty term $\alpha > 0$ is a value to be exogenously fixed for training the model. Given α, the regression model with penalty constraint is then optimized.

In the regularized regression, α is called a hyperparameter of the regression model. Its value is external to the regression model that optimizes the parameters $\hat{b}_0$, $\hat{b}_1$, $\hat{b}_2$, …., $\hat{b}_p$. Hyperparameters arise in many machine learning models such as the number of layers in neural network, the number of branches in a decision tree, the number of trees in a random forest, the

number of clusters in a clustering algorithm, and so on. Given the hyperparameters, the model can then be optimized. Optimizing the model, such as maximizing R^2 or minimizing RMSE etc., can then be done in a second stage by fine-tuning (adjusting) the hyperparameter value to achieve a higher validation R^2 score – this could be done manually or in a programmed grid-step approach, e.g. sklearn.model_selection.GridSearchCV, using the validation data set.

Once the parameters (and chosen or fine-tuned hyperparameter) are determined using the training and validation data sets, the optimization could be repeated given the selected hyperparameter over the combined training and validation data sets. This optimized model with optimal fitted parameters are then used to make predictions in the test data set. High test score R^2 means the model is ready for use in predicting targets based on new cases with given features. If test score R^2 is low, then the model should be improved or changed before using for generalized (new cases) prediction.

In using the Ridge and Lasso regressions on the housing data based on the selected features in our example, we assume the hyperparameter α is fine-tuned at a low 0.05 for Ridge regression and 0.005 for Lasso. In fact, higher α in these regularized regressions yields poor or low R^2 scores.

The training R^2 scores and test R^2 scores for the linear regression with no constraints, and the Ridge and Lasso regressions are tabulated below.

	Training Score R^2	Test Score R^2
Linear Regression (no constraints)	0.62035	0.62137
Ridge Regression (α = 0.05)	0.62019	0.62123
Lasso Regression (α = 0.005)	0.59152	0.59486

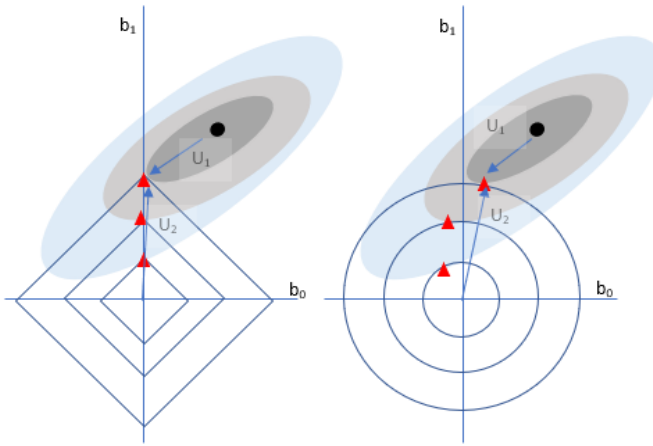
In this case study, the training and test scores for the regularized regressions are a little lower than that of regression without constraints – indicating underfitting when the L^1 , L^2 constraints are imposed on the parameters. Another possible regularized regression is the 'elastic net' regression that is the minimization of objective function as follows, where there are two hyperparameters α_1 and α_2.

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

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 (they add up faster than their squares do). 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, Ridge is not as useful when there is a large number of possibly redundant features to prune instead of just shrinking. Their difference can be illustrated geometrically as follows using a two-dimensional case of minimizing

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

that is quadratic and is also a special case of an ellipse, i.e., an elliptical equation. Under regularization, its minimization is subject to the Lasso type constraints (LHS below) and the Ridge type constraints (RHS below).



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 a larger unconstrained objective function.

U_2 represents the direction of the increase of the respective penalty functions. Given hyperparameter α, the constrained regression optimal solution is a point on the intersection of the two contours (triangles). It is seen that for Lasso, often the optimal points involve zeros in some coefficients.

2.4 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 sub-sample is split into. This procedure improves on the simple split between training and test sets of the data.

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 on the validation/test 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 testing using a validation subset. Thus, there will be k number of test R^2 scores using this k-fold cross-validation approach. The mean and variance of the scores are computed.

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 validation scores.

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

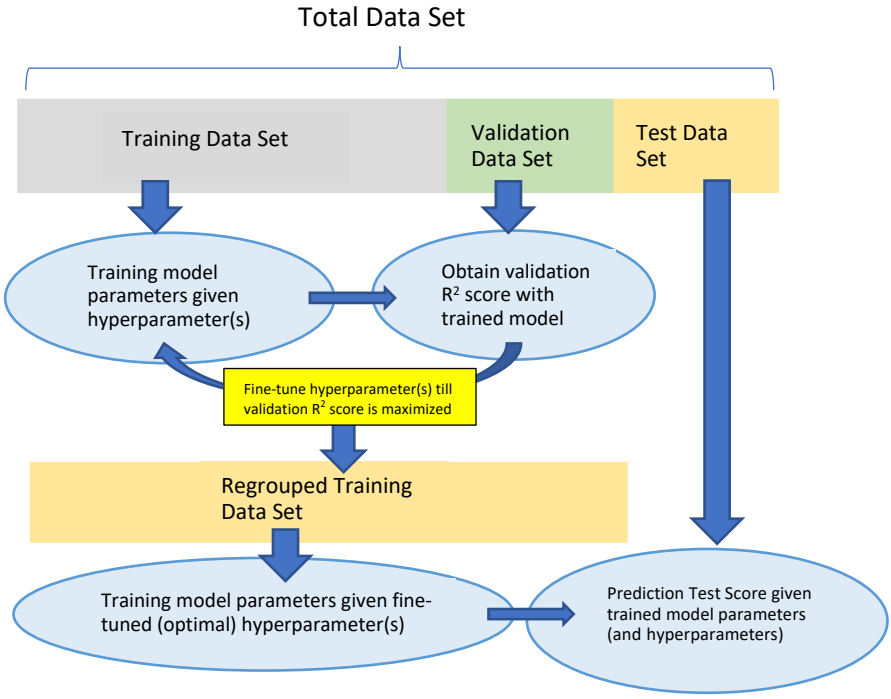


Figure 2.1 (original)

The k-fold cross-validation is a common approach or strategy to improve validation effectiveness and thus come up with an optimal model (with optimized model parameters as well as hyperparameters) to check 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 how hyperparameters can be fine-tuned or optimally selected using this k-fold cross-validation approach.

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. This training data set is then repeated k=5 times in the following arrangement – each repetition, the same training set is divided into k=5 equal folds with k-1 folds serving as training and one-fold serving as validation set. The folds remain unchanged in the repetitions, but the validation set takes a different fold for each repetition. The k-1 folds serve as training set and the prediction is based on their computed parameters applied to the validation set.

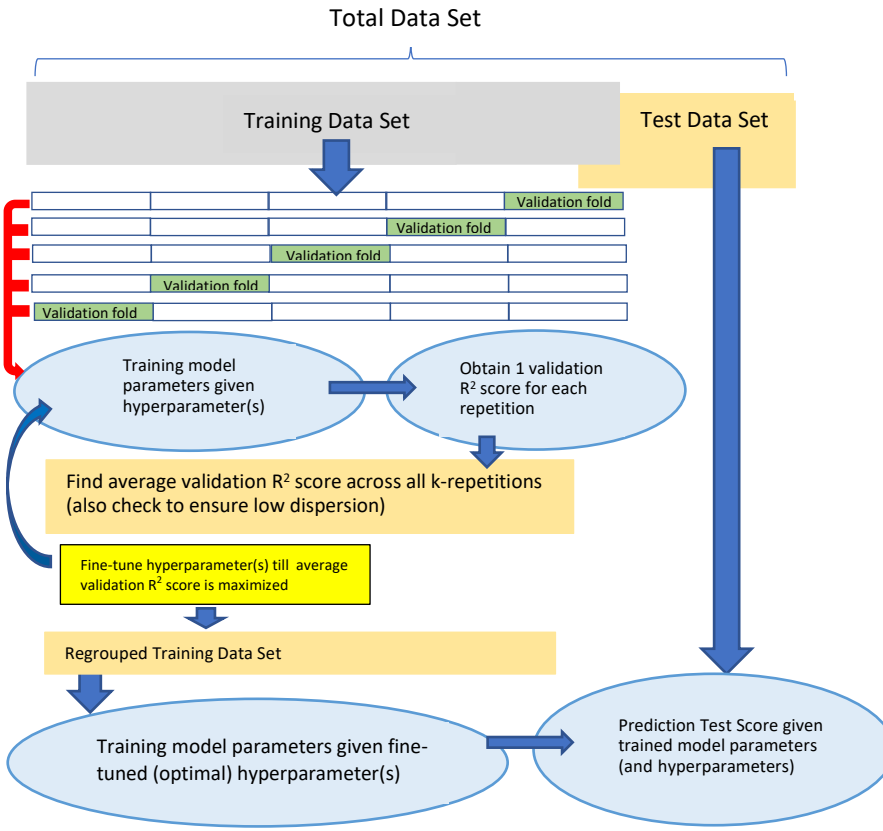


Figure 2.2 (original)

Note that each data point appears only once in the validation data set of all the k repetitions. The k-folds produce 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 [29] shows that the data set is reshuffled anew with a fresh X_train and y_train training set of 15,692 points. Code lines [30] to [32] computes the cross-validation R^2 results of using the process and procedure outlined in Figure 2.2 with k=5 and involving only the original training set of 15,692 points. There are 5 R^2's for each fold. Their average or mean as well as their standard deviation are computed. The validation R^2 score mean and standard deviation (these are not training scores) are reported below the linear regression with no constraints, and the Ridge and Lasso regressions.

	Mean Validation Scores R^2	Standard Deviation of Validation Scores R^2
Linear Regression (no constraints)	0.6178	0.0045
Ridge Regression (α = 0.05)	0.6181	0.0045
Lasso Regression (α = 0.001)	0.6103	0.0073

```
In [29]: ### We can try reshuffling the X_train,y_train point positions here if we wish
         from sklearn.utils import shuffle
         X_shuffle, y_shuffle = shuffle(X_train,y_train, random_state=40)
         ### Rename X_Shuffle y_shuffle
         X_train = X_shuffle
         y_train = y_shuffle
         ### shuffle just randomly permutates the rows

In [30]: ### Now k-fold cross validation is to be performed on X_train y_train reshuflled dataset as in [29], leaving test set intact
         ### The training set here de facto combines a training and validation data sets
         ### Predicting Cross Validation Score
         ### By default cross_val_score uses the scoring provided in the given estimator, which is r2 score here
         ### 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_train, y = y_train, cv = 5)
         ### Note the combined training set is 80% as in [18], randomized by seed = random_state number
         ### 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("%0.4f mean R2 with a standard deviation of %0.4f" % (scoresLinreg.mean(), scoresLinreg.std()))

         [0.6103278  0.61590082 0.6233678  0.61861227 0.62083429]
         0.6178 mean R2 with a standard deviation of 0.0045
```

In [31]:
```
### Now k-fold cross validation is to be performed on X_train y_train dataset in [29], leaving test set intact
### The training set here de facto combines a training and validation data sets
### Predicting Cross Validation Score
### By default cross_val_score uses the scoring provided in the given estimator, which is r2 score here

from sklearn import linear_model
Ridge=linear_model.Ridge(alpha=0.05,)
### Do not need to specify Ridge.fit(X_train, y_train)
### training the algorithm, note regression is done without normalizing the X, y (X already scaled in [17])
### Above steps may be repeated in order to try different hyperparameter of alpha and select one with highest ave R2

from sklearn.model_selection import cross_val_score
scoresRidge = cross_val_score(estimator = Ridge, X = X_train, y = y_train, cv = 5)
### Note the combined training set is 80% as in [18], randomized by seed = random_state number
### 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(scoresRidge)
### Score is R2 measure, there are 5 scores since k=cv=5, one for each repetition
print("%0.4f mean R2 with a standard deviation of %0.4f" % (scoresRidge.mean(), scoresRidge.std()))
```

```
[0.61083575 0.61517579 0.62265187 0.62200811 0.61989958]
0.6181 mean R2 with a standard deviation of 0.0045
```

In [32]:
```
### Now k-fold cross validation is to be performed on X_train y_train reshuffled dataset in [29], leaving test set intact
### The training set here de facto combines a training and validation data sets
### Predicting Cross Validation Score
### By default cross_val_score uses the scoring provided in the given estimator, which is r2 score here

from sklearn import linear_model
Lasso=linear_model.Lasso(alpha=0.001,)
### Do not need to specify Lasso.fit(X_train, y_train)
### training the algorithm, note regression is done without normalizing the X, y
### Above steps may be repeated in order to try different hyperparameter of alpha and select one with highest ave R2

from sklearn.model_selection import cross_val_score
scoresLasso = cross_val_score(estimator = Lasso, X = X_train, y = y_train, cv = 5)
### Note the combined training set is 80% as in [18], randomized by seed = random_state number
### 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(scoresLasso)
### Score is R2 measure, there are 5 scores since k=cv=5, one for each repetition
print("%0.4f mean R2 with a standard deviation of %0.4f" % (scoresLasso.mean(), scoresLasso.std()))
```

```
[0.60127122 0.60612945 0.61319657 0.62278361 0.60809022]
0.6103 mean R2 with a standard deviation of 0.0073
```

As seen in the cross-validation test results using R^2 as the metric or measurement of the accuracy of prediction performance, Ridge performs better than regression without constraints and also better than Lasso in terms of a higher mean R^2 score and lower or same standard deviation in the scores. Lasso does not perform well and it may have to do with under-fitting.

Using the process as in Figure 2.2, different positive values of α can be used to search for the optimal validation proces. The eventual use of the best method, whether linear regression or the regularized regression, together with the optimally selected hyperparameter α, 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.

2.5 Summary

There are clearly advantages in the use of regularized regressions as a comparison to the linear regression model without constraints when the model may have too many features and can be overfitted using the training data.

Cross-validation is a useful strategy to accompany regularized regressions as there is a hyperparameter in the penalty term in the objective function. However, one disadvantage of cross-validation is that it may take up more computing time with the number of repetitions.

Another aspect in machine learning is the scaling of the features so that different scales of different features, e.g., some have huge magnitudes compared with others with miniscule magnitudes, will not affect the performance of the method. This normalizing method can be applied with use of sklearn.preprocessing.StandardScaler (*, copy=True, with_mean = True, with_std=True). The features are then transformed by subtracting the sample mean from each and dividing by the corresponding standard deviation. Or as is done in the exercise in this chapter, a min_max_scaler = preprocessing.MinMaxScaler() is used where each feature is scaled to (X - X.min) / (X.max - X.min), i.e., within (0,1).

When the prediction problem does not have too many features, then using regularizations, particularly Lasso, may yield under-fitting when coefficients are forced to be less impactful. Thus, regularizations should be done only when it improves the fitting and testing accuracies.

There are many studies showing advantages of regularizations – see Jorge Chan-Lau (2017), and Xin and Khalid (2018). As another example, using Kaggle data set on cruise_ship_info, a high predictive accuracy is obtained and improved marginally with regularization.

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.

2.6 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