Unit – IV: Model Evaluation

Generalization Error – Out-of-Sample Evaluation Metrics – Cross Validation – Overfitting – Under Fitting and Model Selection – Prediction by using Ridge Regression – Testing Multiple Parameters by using Grid Search.

Generalization Error

In supervised learning applications in machine learning and statistical learning theory, generalization error (also known as the out-of-sample error) is a measure of how accurately an algorithm is able to predict outcome values for previously unseen data. Notice that the gap between predictions and observed data is induced by model inaccuracy, sampling error, and noise. Some of the errors are reducible but some are not. Choosing the right algorithm and tuning parameters could improve model accuracy, but we will never be able to make our predictions 100% accurate.

Bias-variance decomposition

An important way to understand generalization error is bias-variance decomposition. Intuitively speaking, bias is the error rate in the world of big data. A model has a high bias when, for example, it fails to capture meaningful patterns in the data. Bias is measured by the differences between the expected predicted values and the observed values, in the dataset D when the prediction variables are at the level of x (X=x). In contrast with bias, variance is an algorithm's flexibility to learn patterns in the observed data. Variance is the amount that an algorithm will change if a different dataset is used. A model is of high variance when, for instance, it tries too hard that it not only captures the pattern of meaningful features but also that the meaningless error (overfitting).

Mathematical Notations

D: the training dataset
x: our sample
y: the **real** values of the outcome variable
y_D: the **observed** values of the outcome variable **in dataset D**
$f(x; D)$: the **fitted** values of the outcome variable, i.e. the output of our model when input=x, and the model is learned from dataset D
E_D: the expectation on dataset D
Error(f;D): the generalization error of model f trained on dataset D

Using regression as an example, we have

Generalization error

$$Error(f; D) = E[(f(x; D) - y_D)^2]$$

where

$$\overline{f(x)} = E_D[f(x; D)]$$

bias could be denoted as

$$bias^2(x) = (\overline{f(x)} - y_D)^2$$

Variance be

$$var(x) = E_D[(f(x;D) - \overline{f(x)})^2]$$

And **noise**

$$\epsilon^2 = E_D[(y_D - y)^2]$$

For noise, we have "zero assumption": the expectation of noise is zero.

$$E_D[y_D - y] = 0$$

Now we could decompose generalization error

$$E_D[(f(x;D) - y_D)^2] = E_D[(f(x;D) - \overline{f(x)} + \overline{f(x)} - y_D)^2]$$
$$= E_D[(f(x;D) - \overline{f(x)})^2] + E_D[(\overline{f(x)} - y_D)^2] + 2E_D[(f(x;D) - \overline{f(x)})(\overline{f(x)} - y_D)]$$
$$= E_D[(f(x;D) - \overline{f(x)})^2] + E_D[(\overline{f(x)} - y_D)^2]$$
$$= var(x) + E_D[(\overline{f(x)} - y + y - y_D)^2]$$
$$= var(x) + E_D[(\overline{f(x)} - y)^2] + E_D[(y - y_D)^2] + 2E_D[(\overline{f(x)} - y)(y - y_D)]$$
$$= var(x) + E_D[(\overline{f(x)} - y)^2] + E_D[(y - y_D)^2]$$
$$= var(x) + (\overline{f(x)} - y)^2 + E_D[(y - y_D)^2]$$
$$= var(x) + bias^2(x) + \epsilon^2$$

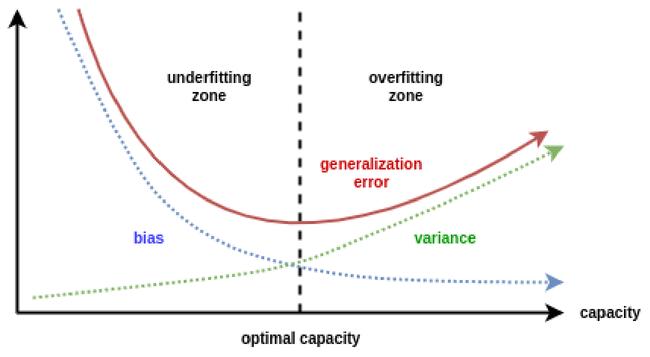
Interpretation

- Bias measures the deviation between the expected output of our model and the real values, so it indicates the fit of our model.
- Variance measures the amount that the outputs of our model will change if a different dataset is used. It is the impacts of using different datasets.
- Noise is the irreducible error, the lowest bound of generalization error for the current task that any model will not be able to get rid of, indicating the difficulty of this task.

These 3 components above determine the model's ability to react to new unseen data rather than just the data that it was trained on.

Bias-Variance Tradeoff

Bias-Variance Tradeoff as a Function of Model Capacity



Generalization error could be measured by MSE. As the model capacity increases, the bias decreases as the model fits the training datasets better. However, the variance increases, as your model become sophisticated to fit more patterns of the current dataset, changing datasets (even if they come from the same distribution) would be impactful. As a data scientist, our challenge lies in finding the optimal capacity — where both bias and variance are low.

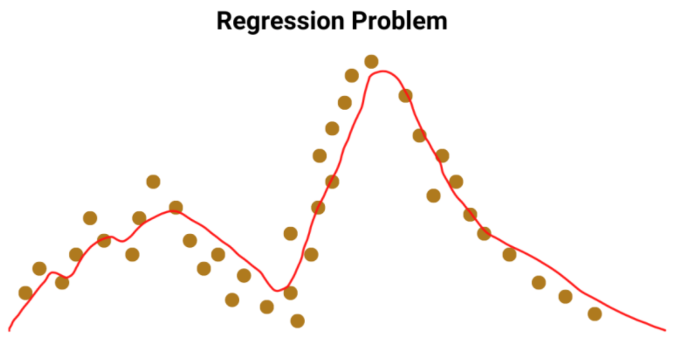
<u>Out-of-Sample Evaluation Metrics</u>

In Regression problems, we map input variables with the continuous output variable(s). For example, predicting the share price in the stock market, predicting atmospheric temperature, etc. Based on the various usabilities, much research is going on in this area to build a more accurate model. When we build a solution for any regression problem, we compare its performance with the existing work. But to compare the two works, there should be some standard metric, like measuring distance in meters, plot size in square feet, etc. Similarly, we need to have some standard evaluation metrics to evaluate two regression models.

Popular methods

1. Mean Absolute Error (MAE)
2. Mean Absolute Percentage Error (MAPE)
3. Mean Squared Error (MSE)
4. Root Mean Squared Error (RMSE)
5. R-Squared (R^2)
6. Adjusted R-Squared (Adjusted-R^2)

But before moving ahead, let's understand one crucial question. Why can't we use accuracy for Regression Problems? Whenever we say that we have built a model, the first question that comes is, "What is the accuracy of our model?". Accuracy is a general term that can be formulated as "Out of all the predictions our model made, how many of them were accurate". As regression problems use supervised data, we know Yactual, and predictions will be considered accurate when Ypredicted is exactly equal to Y_actual. But, in regression problems, we have a continuous target variable. So, if we start evaluating our model on accuracy parameters, we will end up overfitting our model.



To avoid that, we use other evaluation metrics where we consider our model good even if the predictions are very close to the actual value but not exactly equal to the predictions. Hence we can not measure accuracy here. However, to compare the performance of the regression models, there are some defined metrics based on which we can decide which model is performing better. So let's understand these most common metrics for regression problems.

<u>Mean Absolute Error (MAE)</u>

MAE is a fundamental and most used evaluation metric for regression problems. Here we try to calculate the difference between the actual and predicted values. This difference is termed an error. Let's say Ŷi is the predicted value, and Yi is the actual value. So, we can define error in prediction as Error = Yi-Ŷi. This error can either be positive or negative, but we are more concerned about the magnitude. Hence we take modulus, Error = |Yi-Ŷi|.

If we have N such samples present in the data, the total error would be the sum of errors over all those samples, i.e., Total error = Σ |Yi-Ŷi|. But we can not represent the error in terms of total error as the number of samples can be different in different experiments. Hence, we use the mean of this error. Mean says that whenever we will do inference using this model, the value of Ypredicted will lie in the range of (Ypredicted-MAE) ≤ Ypredicted ≤ (Ypredicted+MAE).

$$MAE = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i|$$

Mean Absolute Percentage Error (MAPE)

In different research works, it can be observed that when the target variable feature has a single dimension, some research normalizes that feature, and some don't. For example, suppose our target variable can take values in [0–100]. One method kept the feature as it is, and the second method normalized this feature and brought it in the range of [0,1], where 0 represents 0 and 100 represents 1. But in such a scenario, for the same model, the value of MAE would vary. The error in the first method, where we kept the feature as it is, would be higher than the error in the second method. To take care of these situations, we can define our error in percentage variation from the actual values. In the equation below, Yi is the actual value, Ŷi is the predicted value, and N is the total number of samples.

$$MAPE = \frac{1}{N} \Sigma_i^N \left| \frac{y_i - \hat{y}_i}{y_i} \right| * 100$$

Mean Squared Error (MSE)

MSE is a very popular evaluation metric for regression problems. It is similar to the mean absolute error, but the error is squared here, Error = |Yi-Ŷi|². Similarly, when this squared error is calculated for N samples, the Total Error will beΣ |Yi-Ŷi|². The below formula represent this value as a Mean Squared Error, which reflects the average value of squared error per sample,

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE)

RMSE is the most famous evaluation metric for the regression model. The overall calculation of RMSE is similar to MSE; just the final value is square-rooted as we calculated the square of errors in MSE. We learned in MAE that any new prediction would lie in the range of [Ypredicted-Error, Ypredicted+Error] at the time of inference. In MSE, we squared the error, so we need to calculate the square root to bring it back to the normal stage. That's RMSE for us.

$$RMSE = \sqrt{\frac{1}{N} \Sigma_i^N (y_i - \hat{y}_i)^2}$$

R-Squared (R²)

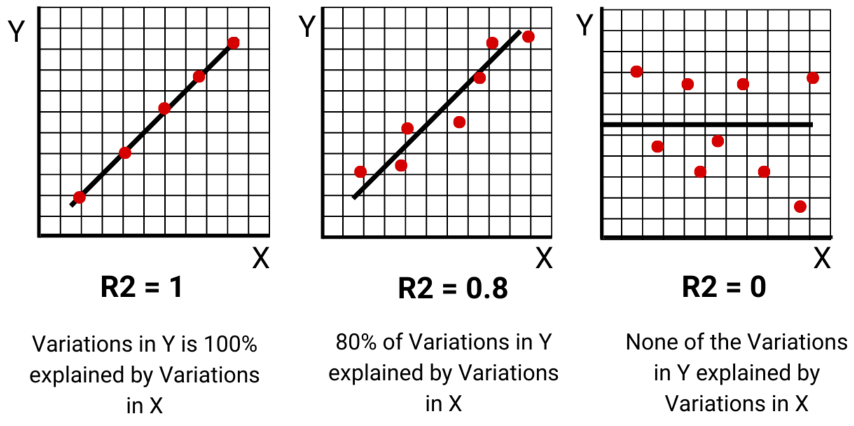
Correlation between two variables explains the strength of their relationship. In contrast, R-squared explains to what extent the variance of one variable explains the variance of the second variable. It is also known as the Coefficient of

Determination. This metric is interesting and important so let's understand it by an example. Suppose we know the salaries of 10 government employees and want to guess the salary for the 11th employee. Assume that we are basic learners and don't have any idea about machine learning techniques. What will be our most reasonable guess?

Mean of salaries of 10 employees, right? The mean value will be considered as the baseline value. We will calculate the baseline error as the squared difference between actual Y and mean value. Let's call this error TSS (Total Sum Squared). Now suppose we know ML, and we built a Machine Learning model to predict the salary. After learning better techniques like ML, we assume that we can improve our prediction capability over the naive guess. Hence the total squared prediction error $\Sigma |Y_i - \hat{Y}_i|^2$ will be lesser than TSS.

R-Square can be calculated using the equation below in which $\overline{y}_i$ is the mean value, and $\hat{Y}_i$ is the predicted value.

$$R^2 = 1 - \frac{\Sigma_i^N (y_i - \hat{y}_i)^2}{\Sigma_i^N (y_i - \bar{y})^2}$$



In theories, the R_squared value will always lie in the range of [0,1], while in practice, values lie in $(-\infty, 1]$. Can you guess when we will have negative R^2? The problem is with the assumption. We thought the ML would beat the performance when we naively guessed the average, but it did not happen. The reason behind negative R^2 can be,

1. Model is not learning the trend that is present in the train data.
2. Too little data has been used to evaluate the model compared to train data.
3. Too many outliers are present in the dataset.

R^2 is a good measure and is widely used in the industry to measure the performance of regression models. But there are serious problems that can misguide machine learning engineers and researchers. If we look carefully, we can change the R^2 value without changing the model at all. Can we guess how? We can increase the input features and make our baseline error higher. If there are too many independent variables, the model can overfit, and R^2 would be high. But on the test data, it will perform poorly.

Adjusted R-Squared (Adjusted R^2)

To tackle R^2's problems, researchers formed a new metric that is considered the improvement in R^2 and is known as adjusted R^2. In the equation below, N is the total number of data samples, and k is the number of independent variables.

$$Adjusted\_R^2 = 1 - [(\frac{N-1}{N-k-1}) * (1 - R^2)]$$

Adjusted R-Squared value will always be lesser than the traditional R-Squared value. Whenever we add a new independent variable, it will affect the calculations. So, we can never be misguided with the score now.

Industries and Research papers are more inclined toward RMSE or MSE values, so we must compare our results with these parameters. Additionally, there is a slight inclination toward R-Squared values as well as it can be directly correlated with the accuracy. Adjusted R^2 is the only parameter considering the overfitting problem.

Cross Validation

Cross-Validation also referred to as out of sampling technique is an essential element of a data science project. It is a resampling procedure used to evaluate machine learning models and access how the model will perform for an independent test dataset. There are 8 different cross-validation techniques having their pros and cons, listed below:

1. Leave p out cross-validation
2. Leave one out cross-validation
3. Holdout cross-validation
4. Repeated random subsampling validation
5. k-fold cross-validation
6. Stratified k-fold cross-validation
7. Time Series cross-validation
8. Nested cross-validation

Why Cross-Validation is Important?

We often randomly split the dataset into train data and test data to develop a machine learning model. The training data is used to train the ML model and the same model is tested on independent testing data to evaluate the performance of the model. With the change in the random state of the split, the accuracy of the model also changes, so we are not able to achieve a fixed accuracy for the model. The testing data should be kept independent of the training data so that no data leakage occurs. During the development of an ML model using the training data, the model performance needs to be evaluated. Here's the importance of cross-validation data comes into the picture.

Data needs to split into:

- Training data: Used for model development
- Validation data: Used for validating the performance of the same model

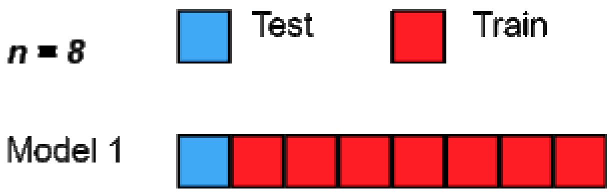
In simple terms cross-validation allows us to utilize our data even better. You can further read, working, and implementation of 7 types of Cross-Validation techniques.

1. Leave p-out cross-validation:

Leave p-out cross-validation (LpOCV) is an exhaustive cross-validation technique, that involves using p-observation as validation data, and remaining data is used to train the model. This is repeated in all ways to cut the original sample on a validation set of p observations and a training set. A variant of LpOCV with p=2 known as leave-pair-out cross-validation has been recommended as a nearly unbiased method for estimating the area under ROC curve of a binary classifier.

2. Leave-one-out cross-validation:

Leave-one-out cross-validation (LOOCV) is an exhaustive cross-validation technique. It is a category of LpOCV with the case of p=1.



For a dataset having n rows, 1st row is selected for validation, and the rest (n-1) rows are used to train the model. For the next iteration, the 2nd row is selected for validation and rest to train the model. Similarly, the process is repeated until n steps or the desired number of operations. Both the above two cross-validation techniques are the types of exhaustive cross-validation. Exhaustive cross-validation methods are cross-validation methods that learn and test in all possible ways. They have the same pros and cons discussed below:

Pros:

- Simple, easy to understand, and implement.

Cons:

- The model may lead to a low bias.
- The computation time required is high.

3. Holdout cross-validation:

The holdout technique is an exhaustive cross-validation method, that randomly splits the dataset into train and test data depending on data analysis. In the case of holdout cross-validation, the dataset is randomly split into training and validation data. Generally, the split of training data is more than test data. The training data is used to induce the model and validation data is evaluates the performance of the model. The more data is used to train the model, the better the model is. For the holdout cross-validation method, a good amount of data is isolated from training.

Pros:

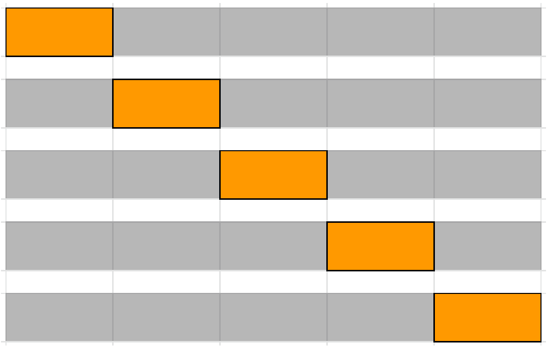
- Same as previous.

Cons:

- Not suitable for an imbalanced dataset.
- A lot of data is isolated from training the model.

4. k-fold cross-validation:

In k-fold cross-validation, the original dataset is equally partitioned into k subparts or folds. Out of the k-folds or groups, for each iteration, one group is selected as validation data, and the remaining (k-1) groups are selected as training data. The process is repeated for k times until each group is treated as validation and remaining as training data.



The final accuracy of the model is computed by taking the mean accuracy of the k-models validation data.

$$\mathrm{acc}_{\mathrm{cv}} = \sum_{i=1}^{k} \frac{\mathrm{acc}_i}{k}$$

LOOCV is a variant of k-fold cross-validation where k=n.

Pros:

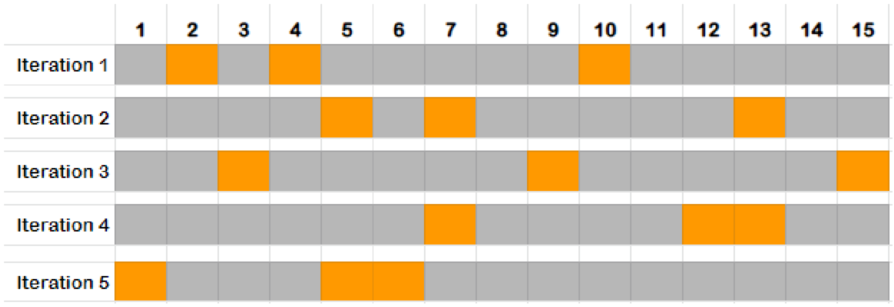
- The model has low bias
- Low time complexity
- The entire dataset is utilized for both training and validation.

Cons:

- Not suitable for an imbalanced dataset.

5. Repeated random subsampling validation

Repeated random subsampling validation also referred to as Monte Carlo cross-validation splits the dataset randomly into training and validation. Unlikely k-fold cross-validation split of the dataset into not in groups or folds but splits in this case in random. The number of iterations is not fixed and decided by analysis. The results are then averaged over the splits.



$$\mathrm{acc}_{\mathrm{cv}} = \sum_{i=1}^{k} \frac{\mathrm{acc}_i}{k}$$

Pros:

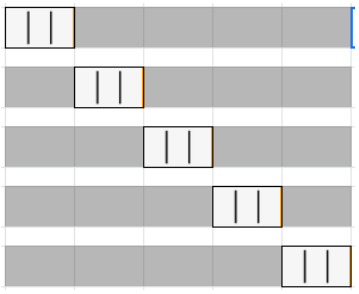
- The proportion of train and validation splits is not dependent on the number of iterations or partitions.

Cons:

- Some samples may not be selected for either training or validation.
- Not suitable for an imbalanced dataset.

6. Stratified k-fold cross-validation:

For all the cross-validation techniques discussed above, they may not work well with an imbalanced dataset. Stratified k-fold cross-validation solved the problem of an imbalanced dataset. In Stratified k-fold cross-validation, the dataset is partitioned into k groups or folds such that the validation data has an equal number of instances of target class label. This ensures that one particular class is not over present in the validation or train data especially when the dataset is imbalanced.



Stratified k-fold cross-validation, Each fold has equal instances of the target class. The final score is computed by taking the mean of scores of each fold.

Pros:

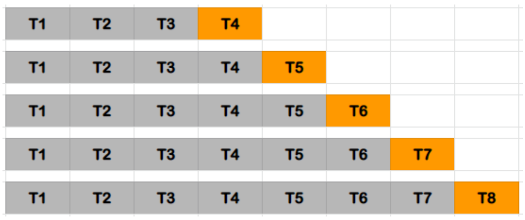
- Works well for an imbalanced dataset.

Cons:

- Now suitable for time series dataset.

7. Time Series cross-validation:

The order of the data is very important for time-series related problem. For time-related dataset random split or k-fold split of data into train and validation may not yield good results. For the time-series dataset, the split of data into train and validation is according to the time also referred to as forward chaining method or rolling cross-validation. For a particular iteration, the next instance of train data can be treated as validation data.



As mentioned in the above diagram, for the 1st iteration, 1st 3 rows are considered as train data and the next instance T4 is validation data. The chance of choice of train and validation data is forwarded for further iterations.

8. Nested cross-validation:

In the case of k-fold and stratified k-fold cross-validation, we get a poor estimate of the error in training and test data. Hyperparameter tuning is done separately in the earlier methods. When cross-validation is used simultaneously for tuning the hyperparameters and generalizing the error estimate, nested cross-validation is required. Nested Cross Validation can be applicable in both k-fold and stratified k-fold variants. Read the below article to know more about nested cross-validation and its implementation:

Limitations of Cross-Validation

- For the ideal conditions, it provides the optimum output. But for the inconsistent data, it may produce a drastic result. So, it is one of the big disadvantages of cross-validation, as there is no certainty of the type of data in machine learning.
- In predictive modeling, the data evolves over a period, due to which, it may face the differences between the training set and validation sets. Such as if we create a model for the prediction of stock market values, and the data is trained on the previous 5 years stock values, but the realistic future values for the next 5 years may drastically different, so it is difficult to expect the correct output for such situations.

Applications of Cross-Validation

- This technique can be used to compare the performance of different predictive modeling methods.
- It has great scope in the medical research field.
- It can also be used for the meta-analysis, as it is already being used by the data scientists in the field of medical statistics.

Cross-validation is used to compare and evaluate the performance of ML models. k-fold and stratified k-fold cross-validations are the most used techniques. Time series cross-validation works best with time series related problems.

Overfitting – Under Fitting and Model Selection

Overfitting and Underfitting are the two main problems that occur in machine learning and degrade the performance of the machine learning models. The main goal of each machine learning model is to generalize well. Here generalization defines the ability of an ML model to provide a suitable output by adapting the given set of unknown input. It means after providing training on the dataset, it can produce reliable and accurate output. Hence, the underfitting and overfitting are the two terms that need to be checked for the performance of the model and whether the model is generalizing well or not. Before understanding the overfitting and underfitting, let's understand some basic term that will help to understand this topic well:

- Signal: It refers to the true underlying pattern of the data that helps the machine learning model to learn from the data.
- Noise: Noise is unnecessary and irrelevant data that reduces the performance of the model.
- Bias: Bias is a prediction error that is introduced in the model due to oversimplifying the machine learning algorithms. Or it is the difference between the predicted values and the actual values.
- Variance: If the machine learning model performs well with the training dataset, but does not perform well with the test dataset, then variance occurs.

Overfitting

Overfitting occurs when our machine learning model tries to cover all the data points or more than the required data points present in the given dataset. Because of this, the model starts caching noise and inaccurate values present in the dataset, and all these factors reduce the efficiency and accuracy of the model. The overfitted model has low bias and high variance. The chances of occurrence of overfitting increase as much we provide training to our model. It means the more we train our model, the more chances of occurring the overfitted model. Overfitting is the main problem that occurs in supervised learning.

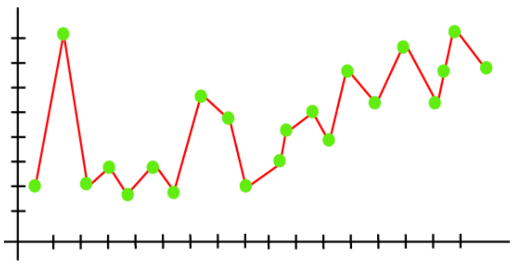
Reasons for Overfitting are as follows:

- High variance and low bias
- The model is too complex
- The size of the training data

Techniques to reduce overfitting:

- Increase training data.
- Reduce model complexity.
- Early stopping during the training phase (have an eye over the loss over the training period as soon as loss begins to increase stop training).
- Ridge Regularization and Lasso Regularization
- Use dropout for neural networks to tackle overfitting.

Example: The concept of the overfitting can be understood by the below graph of the linear regression output:



As we can see from the above graph, the model tries to cover all the data points present in the scatter plot. It may look efficient, but in reality, it is not so. Because the goal of the regression model to find the best fit line, but here we have not got any best fit, so, it will generate the prediction errors.

How to avoid the Overfitting in Model

Both overfitting and underfitting cause the degraded performance of the machine learning model. But the main cause is overfitting, so there are some ways by which we can reduce the occurrence of overfitting in our model.

- Cross-Validation
- Training with more data
- Removing features
- Early stopping the training
- Regularization
- Ensembling

<u>Underfitting</u>

Underfitting occurs when our machine learning model is not able to capture the underlying trend of the data. To avoid the overfitting in the model, the fed of training data can be stopped at an early stage, due to which the model may not learn enough from the training data. As a result, it may fail to find the best fit of the dominant trend in the data. In the case of underfitting, the model is not able to learn enough from the training data, and hence it reduces the accuracy and produces unreliable predictions. An underfitted model has high bias and low variance.

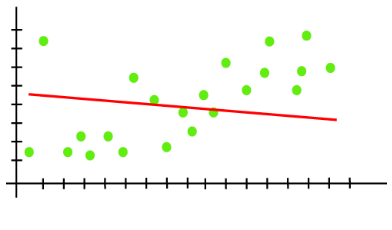
Reasons for Underfitting:

- High bias and low variance
- The size of the training dataset used is not enough.
- The model is too simple.
- Training data is not cleaned and also contains noise in it.

Techniques to reduce underfitting:

- Increase model complexity
- Increase the number of features, performing feature engineering
- Remove noise from the data.
- Increase the number of epochs or increase the duration of training to get better results.

Example: We can understand the underfitting using below output of the linear regression model:



As we can see from the above diagram, the model is unable to capture the data points present in the plot.

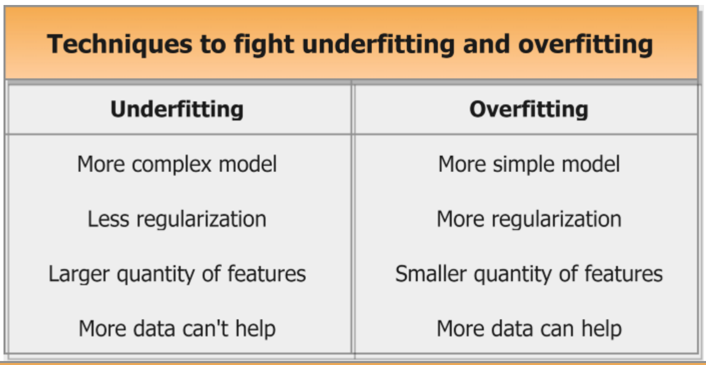
How to avoid underfitting:

- By increasing the training time of the model.
- By increasing the number of features.

Goodness of Fit

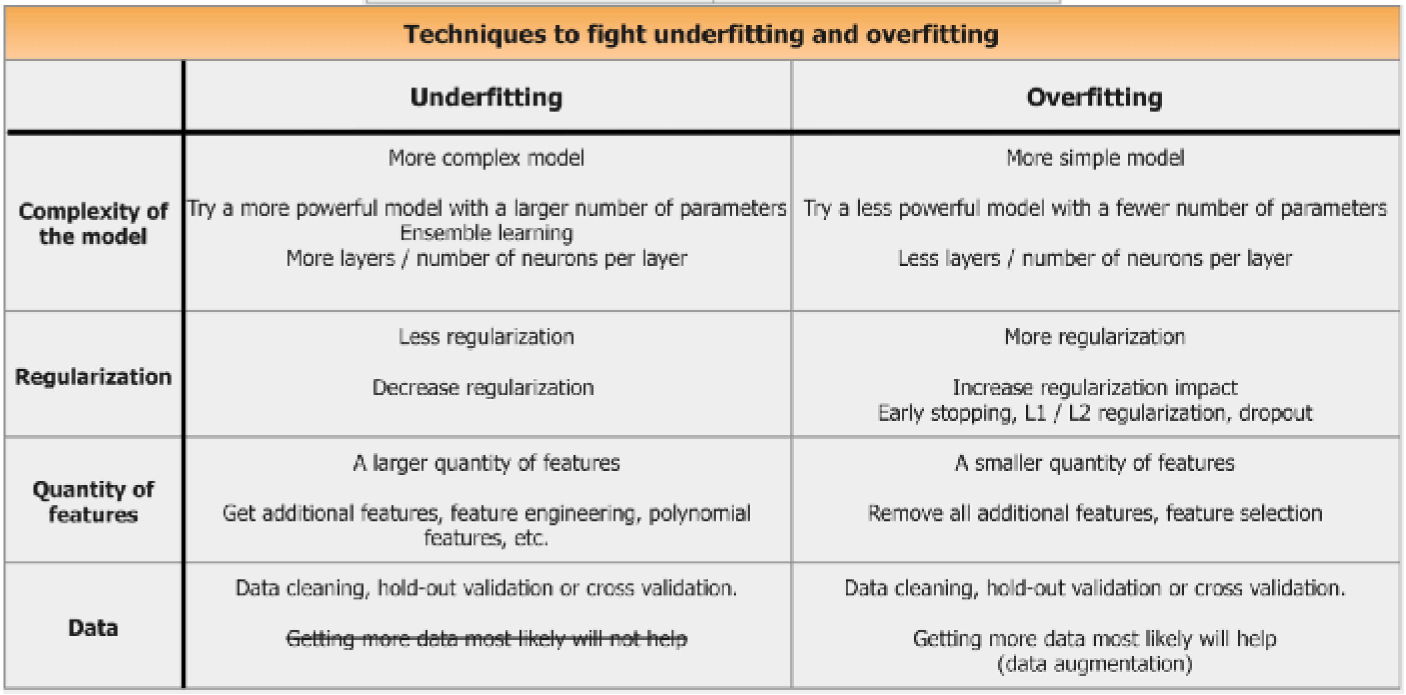
The "Goodness of fit" term is taken from the statistics, and the goal of the machine learning models to achieve the goodness of fit. In statistics modeling, it defines how closely the result or predicted values match the true values of the dataset. The model with a good fit is between the underfitted and overfitted model, and ideally, it makes predictions with 0 errors, but in practice, it is difficult to achieve it. As when we train our model for a time, the errors in the training data go down, and the same happens with test data. But if we train the model for a long duration, then the performance of the model may decrease due to the overfitting, as the model also learn the noise present in the dataset. The errors in the test dataset start increasing, so the point, just before the raising of errors, is the good point, and we can stop here for achieving a good model. There are two other methods by which we can get a good point for our model, which are the resampling method to estimate model accuracy and validation dataset.

Good Fit in a Statistical Model: Ideally, the case when the model makes the predictions with 0 error, is said to have a good fit on the data. This situation is achievable at a spot between overfitting and underfitting. In order to understand it, we will have to look at the performance of our model with the passage of time, while it is learning from the training dataset. With the passage of time, our model will keep on learning, and thus the error for the model on the training and testing data will keep on decreasing. If it will learn for too long, the model will become more prone to overfitting due to the presence of noise and less useful details. Hence the performance of our model will decrease. In order to get a good fit, we will stop at a point just before where the error starts increasing. At this point, the model is said to have good skills in training datasets as well as our unseen testing dataset.



Techniques to fight underfitting and overfitting

Underfitting	Overfitting
More complex model	More simple model
Less regularization	More regularization
Larger quantity of features	Smaller quantity of features
More data can't help	More data can help



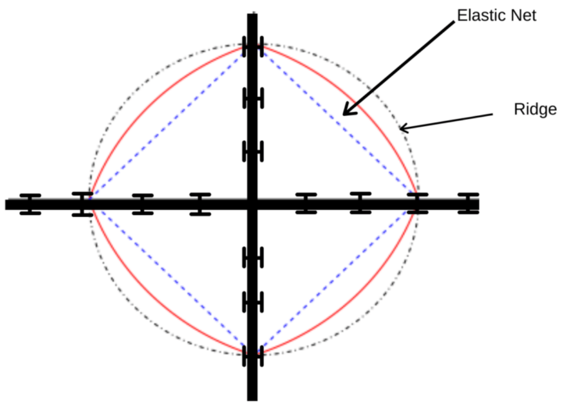
Techniques to fight underfitting and overfitting

	Underfitting	Overfitting
Complexity of the model	More complex model Try a more powerful model with a larger number of parameters Ensemble learning More layers / number of neurons per layer	More simple model Try a less powerful model with a fewer number of parameters Less layers / number of neurons per layer
Regularization	Less regularization Decrease regularization	More regularization Increase regularization impact Early stopping, L1 / L2 regularization, dropout
Quantity of features	A larger quantity of features Get additional features, feature engineering, polynomial features, etc.	A smaller quantity of features Remove all additional features, feature selection
Data	Data cleaning, hold-out validation or cross validation. ~~Getting more data most likely will not help~~	Data cleaning, hold-out validation or cross validation. Getting more data most likely will help (data augmentation)

Some tools and techniques have not been covered in this article. For example, I consider data cleaning and cross-validation or hold-out validation to be common practices in any machine learning project, but they can also be considered as tools to combat overfitting. You may notice that to eliminate underfitting or overfitting, you need to apply diametrically opposite actions. So if you initially "misdiagnosed" your model, you can spend a lot of time and money on empty work (for example, getting new data when in fact you need to complicate the model). That's why it is so important — hours of analysis can save your days and weeks of work. In addition to the usual analysis of the model quality (train/test errors), there are many techniques for understanding exactly what needs to be done to improve the model (error analysis, ceiling analysis, etc.). However, all these procedures have the purpose of understanding where to move and what to pay attention to.

Prediction by using Ridge Regression

Ridge regression is the method used for the analysis of multicollinearity in multiple regression data. It is most suitable when a data set contains a higher number of predictor variables than the number of observations. The second-best scenario is when multicollinearity is experienced in a set.



Multicollinearity happens when predictor variables exhibit a correlation among themselves. Ridge regression aims at reducing the standard error by adding some bias in the estimates of the regression. The reduction of the standard error in regression estimates significantly increases the reliability of the estimates.

- Ridge regression is a technique used to eliminate multicollinearity in data models.
- In a case where observations are fewer than predictor variables, ridge regression is the most appropriate technique.
- Ridge regression constraint variables form a circular shape when plotted, unlike the LASSO plot, which forms a diamond shape.

Variables Standardization in Ridge Regression

Variables standardization is the initial procedure in ridge regression. Both the independent and dependent variables require standardization through subtraction of their averages and a division of the result with the standard deviations. It is common practice to annotate in a formula whether the variables therein are standardized or not. Therefore, all ridge regression computations use standardized variables to avoid the notations on whether individual variables have been standardized. The coefficients can then be reverted to their original scales in the end.

Ridge Regression vs. Least Squares

Ridge regression is a better predictor than least squares regression when the predictor variables are more than the observations. The least squares method cannot tell the difference between more useful and less useful predictor variables and includes all the predictors while developing a model. This reduces the accuracy of the model, resulting in overfitting and redundancy. All of the above challenges are addressed by ridge regression. Ridge regression works with the advantage of not requiring unbiased estimators – rather, it adds bias to estimators to reduce the standard error. It adds bias enough to make the estimates a reliable representation of the population of data.

Shrinkage and Regularization

A ridge estimator is a shrinkage tool used in ridge regression. A shrinkage estimator is a parameter that produces new estimators that have been shrunk to give a value closer to the real population parameters. A least squares estimate can be shrunk using a ridge estimator to improve the estimate, especially when there is multicollinearity in the data. Regularization in ridge regression includes the application of a penalty to coefficients. The shrinkage involves the application of the same factor on the coefficients. This means that no coefficient will be left out when building the model.

Multicollinearity

Multicollinearity is the existence of a correlation between independent variables in modeled data. It can cause inaccuracy in the regression coefficient estimates. It can also magnify the standard errors in the regression coefficients and reduce the efficiency of any t-tests. It can produce deceiving results and p-values and increase the redundancy of a model, making its predictability inefficient and less reliable. Multicollinearity can be introduced into the data from various sources, such as during data collection, from the population or linear model constraints, or an over-defined model, outliers, or model specification or choice.

Data collection may cause multicollinearity when it is sourced using an inappropriate sampling procedure. The data may come from a smaller subset than expected – hence, the effect. Population or model constraints cause multicollinearity due to physical, legal, or political constraints, which are natural, regardless of the type of sampling method used. Over-defining a model will also cause multicollinearity due to the existence of more variables than observations. It is avoidable during the development of a model. The model's choice or specification can also cause multicollinearity due to the use of independent variables previously interacting in the initial variable set. Outliers are extreme variable values that can cause multicollinearity. The multicollinearity can be reversed by the elimination of the outliers before applying ridge regression.

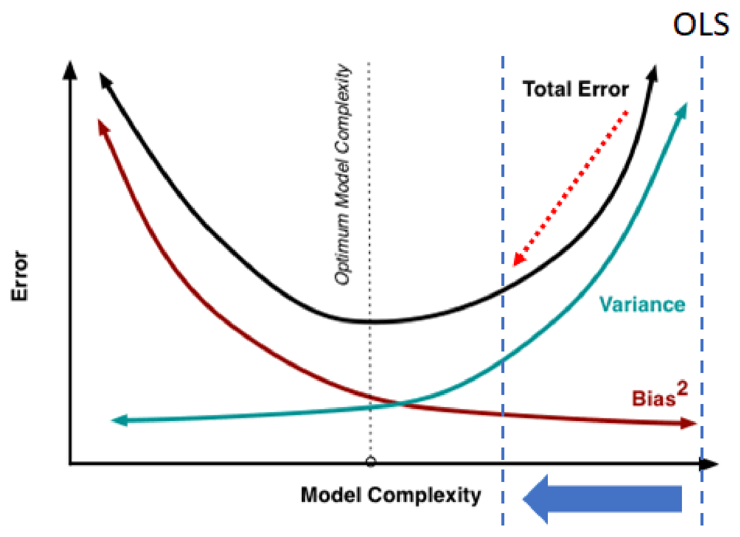
Multicollinearity Detection and Correction

The detection of multicollinearity is key to the reduction of standard errors in models for predictability efficiency. First, one can detect by investigating independent variables for correlation in pairwise scatter plots. High pairwise correlations of independent variables can mean the presence of multicollinearity. Secondly, one can detect multicollinearity through the consideration of Variance Inflation Factors (VIFs). A VIF score of 10 or more shows that variables are collinear. Thirdly, one can detect multicollinearity by checking whether the correlation matrix eigenvalues are close to zero. One should use the condition numbers, as opposed to using the eigenvalue numerical sizes. The larger the condition numbers, the more the multicollinearity.

Multicollinearity correction depends on the cause. When the source of collinearity is data collection, for example, the correction will involve collecting additional data from the proper subpopulation. If the cause is the linear model choice, the correction will include simplifying the model by the proper variable selection methods. If the causes of multicollinearity are certain observations, eliminate the observations. Ridge regression is also an effective eliminator of multicollinearity.

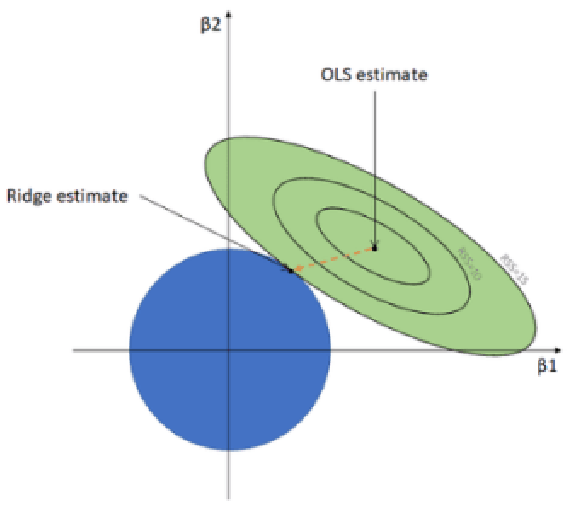
Use of Ridge Regression

We know that the Ordinary Least Square Method (OLS) treats all the variables in an unbiased manner. So, as more variables are incorporated, the OLS model becomes more complicated. The OLS model is on the right side with the low bias and high variance in the figure below. The OLS model's position is stationary and fixed, but the change in position can occur when ridge regression comes into play. In ridge regression, the model coefficients will change as we tune the lambda parameter.



<u>Geometric Understanding of Ridge Regression</u>

The figure below represents the geometric interpretation to compare OLS and ridge regression. Each contour is connected to spots where the RSS is the same, centered with the OLS estimate, where the RSS is the lowest. Also, the OLS estimate is the point where it best fits the training set (low-bias).



Ordinary Least Squares

Vectorized Version - The vector norm is nothing but the following definition.

$$\|B\|_2 = \sqrt{\beta_0^2 + \beta_1^2 + \cdots + \beta_p^2}$$

The subscript '2' is as in 'L2 norm'. We only care about the L2 norm at this moment, so we can construct the equation we've already seen.

Mathematical Notation

In the following equation, the first term is OLS and the second term with the lambda parameter makes ridge regression.

$$\hat{\beta}^{ridge} = \underset{\beta \in \mathbb{R}}{argmin} \|y - XB\|_2^2 + \lambda \|B\|_2^2$$

Having the lambda parameter is often called "penalty," as it causes a significant RSS increase. We try and iterate certain values onto lambda and then evaluate the model with a measurement like Mean Square Error (MSE). The value of the lambda, which minimizes MSE, is selected as the final one. The ridge regression model is far better than the OLS model in prediction. In the formula below, if lambda is equal to zero i-e no penalty, ridge regression becomes the same as OLS.

$$\hat{\beta}^{ridge} = (X^T X + \lambda I)^{-1} X^T Y$$

Model Hyperparameter Optimization

Machine learning models have hyperparameters. Hyperparameters are points of choice or configuration that allow a machine learning model to be customized for a specific task or dataset.

Hyperparameter: Model configuration argument specified by the developer to guide the learning process for a specific dataset. Machine learning models also have parameters, which are the internal coefficients set by training or optimizing the model on a training dataset. Parameters are different from hyperparameters. Parameters are learned automatically; hyperparameters are set manually to help guide the learning process.

Difference Between a Parameter and a Hyperparameter

Typically a hyperparameter has a known effect on a model in the general sense, but it is not clear how to best set a hyperparameter for a given dataset. Further, many machine learning models have a range of hyperparameters and they may interact in nonlinear ways. As such, it is often required to search for a set of hyperparameters that result in the best performance of a model on a dataset. This is called hyperparameter optimization, hyperparameter tuning, or hyperparameter search. An optimization procedure involves defining a search space. This can be thought of geometrically as an n-dimensional volume, where each hyperparameter represents a different dimension and the scale of the dimension are the values that the hyperparameter may take on, such as real-valued, integer-valued, or categorical.

Search Space

Volume to be searched where each dimension represents a hyperparameter and each point represents one model configuration. A point in the search space is a vector with a specific value for each hyperparameter value. The goal of the optimization procedure is to find a vector that results in the best performance of the model after learning, such as maximum accuracy or minimum error. A range of different optimization algorithms may be used, although two of the simplest and most common methods are random search and grid search.

- Random Search. Define a search space as a bounded domain of hyperparameter values and randomly sample points in that domain.
- Grid Search. Define a search space as a grid of hyperparameter values and evaluate every position in the grid.

Grid search is great for spot-checking combinations that are known to perform well generally. Random search is great for discovery and getting hyperparameter combinations that you would not have guessed intuitively, although it often requires more time to execute. More advanced methods are sometimes used, such as Bayesian Optimization and Evolutionary Optimization.

Hyperparameter Optimization for Regression

In this section we will use hyper optimization to discover a top-performing model configuration for the auto insurance dataset. The auto insurance dataset is a standard machine learning dataset comprising 63 rows of data with 1 numerical input variable and a numerical target variable. Using a test harness of repeated stratified 10-fold cross-validation with 3 repeats, a naive model can achieve a mean absolute error (MAE) of about 66. A top performing model can achieve a MAE on this same test harness of about 28. This provides the bounds of expected performance on this dataset. The dataset involves predicting the total amount in claims (thousands of Swedish Kronor) given the number of claims for different geographical regions.

The example below downloads the dataset and summarizes its shape.

```
# summarize the auto insurance dataset
from pandas import read_csv
# load dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/auto-insurance.csv'
dataframe = read_csv(url, header=None)
# split into input and output elements
data = dataframe.values
X, y = data[:, :-1], data[:, -1]
print(X.shape, y.shape)
```

Running the example downloads the dataset and splits it into input and output elements. As expected, we can see that there are 63 rows of data with 1 input variable.

```
(63, 1) (63,)
```

Next, we can use hyperparameter optimization to find a good model configuration for the auto insurance dataset. To keep things simple, we will focus on a linear model, the linear regression model and the common hyperparameters tuned for this model.

The need for hyperparameter tuning

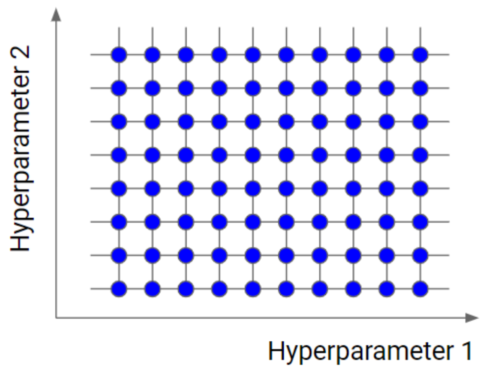
Hyperparameters are model parameters whose values are set before training. For example, the number of neurons of a feed-forward neural network is a hyperparameter, because we set it before training. Another example of hyperparameter is the number of trees in a random forest or the penalty intensity of a Lasso regression. They are all numbers that are set before the training phase and their values affect the behavior of the model.

Why should we tune the hyperparameters of a model? Because we don't really know their optimal values in advance. A model with different hyperparameters is, actually, a different model so it may have a lower performance. In the case of neural networks, a low number of neurons could lead to underfitting and a high number could lead to overfitting. In both cases, the model is not good, so we need to find the intermediate number of neurons that leads to the best performance.

If the model has several hyperparameters, we need to find the best combination of values of the hyperparameters searching in a multi-dimensional space. That's why hyperparameter tuning, which is the process of finding the right values of the hyperparameters, is a very complex and time-expensive task. Let's see two of the most important algorithms for hyperparameter tuning, that are grid search and random search.

Grid search

Grid search is the simplest algorithm for hyperparameter tuning. Basically, we divide the domain of the hyperparameters into a discrete grid. Then, we try every combination of values of this grid, calculating some performance metrics using cross-validation. The point of the grid that maximizes the average value in cross-validation, is the optimal combination of values for the hyperparameters.

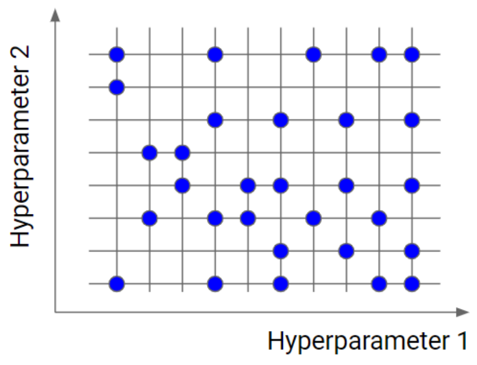


Example of a grid search

Grid search is an exhaustive algorithm that spans all the combinations, so it can actually find the best point in the domain. The great drawback is that it's very slow. Checking every combination of the space requires a lot of time that, sometimes, is not available. Don't forget that every point in the grid needs k-fold cross-validation, which requires k training steps. So, tuning the hyperparameters of a model in this way can be quite complex and expensive. However, if we look for the best combination of values of the hyperparameters, grid search is a very good idea.

Random search

Random search is similar to grid search, but instead of using all the points in the grid, it tests only a randomly selected subset of these points. The smaller this subset, the faster but less accurate the optimization. The larger this dataset, the more accurate the optimization but the closer to a grid search.



Example of random search

Random search is a very useful option when you have several hyperparameters with a fine-grained grid of values. Using a subset made by 5-100 randomly selected points, we are able to get a reasonably good set of values of the hyperparameters. It will not likely be the best point, but it can still be a good set of values that gives us a good model.

Grid Search for Regression

As a grid search, we cannot define a distribution to sample and instead must define a discrete grid of hyperparameter values. As such, we will specify the "alpha" argument as a range of values on a log-10 scale.

```
...
# define search space
space = dict()
space['solver'] = ['svd', 'cholesky', 'lsqr', 'sag']
space['alpha'] = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100]
space['fit_intercept'] = [True, False]
```

```
space['normalize'] = [True, False]
```

Grid search for regression requires that the "scoring" be specified, much as we did for random search. In this case, we will again use the negative MAE scoring function.

```
...
# define search
search = GridSearchCV(model, space, scoring='neg_mean_absolute_error', n_jobs=-1, cv=cv)
```

Tying this together, the complete example of grid searching linear regression configurations for the auto insurance dataset is listed below.

```
# grid search linear regression model on the auto insurance dataset
from pandas import read_csv
from sklearn.linear_model import Ridge
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import GridSearchCV
# load dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/auto-insurance.csv'
dataframe = read_csv(url, header=None)
# split into input and output elements
data = dataframe.values
X, y = data[:, :-1], data[:, -1]
# define model
model = Ridge()
# define evaluation
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
# define search space
space = dict()
space['solver'] = ['svd', 'cholesky', 'lsqr', 'sag']
space['alpha'] = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100]
space['fit_intercept'] = [True, False]
space['normalize'] = [True, False]
# define search
search = GridSearchCV(model, space, scoring='neg_mean_absolute_error', n_jobs=-1, cv=cv)
# execute search
result = search.fit(X, y)
# summarize result
print('Best Score: %s' % result.best_score_)
print('Best Hyperparameters: %s' % result.best_params_)
```

Running the example may take a minute. It is fast because we are using a small search space and a fast model to fit and evaluate. Again, you may see some warnings during the optimization for invalid configuration combinations. These can be safely ignored. At the end of the run, the best score and hyperparameter configuration that achieved the best performance are reported. Your specific results will vary given the stochastic nature of the optimization procedure. Try running the example a few times. In this case, we can see that the best configuration achieved a MAE of about 29.2, which is nearly identical to what we achieved with the random search in the previous section. Interestingly, the hyperparameters are also nearly identical, which is good confirmation.

```
Best Score: -29.275708614337326
Best Hyperparameters: {'alpha': 0.1, 'fit_intercept': True, 'normalize': False, 'solver': 'sag'}
```

<u>How to Choose Hyperparameters to Search?</u>

Most algorithms have a subset of hyperparameters that have the most influence over the search procedure. These are listed in most descriptions of the algorithm. Tune Hyperparameters for Classification Machine Learning Algorithms If you are unsure:

- Review papers that use the algorithm to get ideas.

- Review the API and algorithm documentation to get ideas.
- Search all hyperparameters.

How to Use Best-Performing Hyperparameters?

- Define a new model and set the hyperparameter values of the model to the values found by the search.
- Then fit the model on all available data and use the model to start making predictions on new data.
- This is called preparing a final model.