Machine Learning

LVC 2: Model Evaluation, Cross-Validation and Bootstrapping

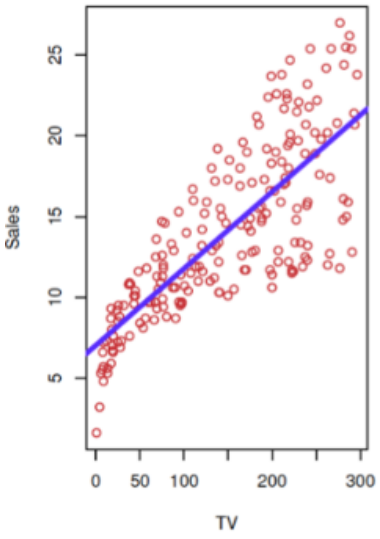
In regression problems, **linear regression** has been one of the prominent algorithms to give an appropriate model for correct predictions. As it is a simple algorithm to use and make predictions, it is also one of the preferred algorithms to tackle regression problems. Once we build the linear regression model, the next step is to interpret the results.

Let's start by disusssing **what can go wrong** with the model that may lead to non-reliable results or misinterpret the results.

Heteroscedasticity

To get a better fit of the model, it is required to have a **constant variance of residuals** along the best-fit line. If the variance of residuals along the line of regression is varying, this case is called **heteroscedasticity**. The formulas in regression assume that all residuals are drawn from a population that has a constant variance (homoscedasticity). Hence, if the assumption is violated, then the results of linear regression might not be reliable.

The below graph shows that the variance of residuals is not constant along the regression line for simple regression using the TV feature. The residuals have a larger variance for larger values of TV, or in other words, the variance of residuals is not constant.



In the above figure, we used the scatter plot to detect heteroscedasticity, but it might not be feasible to do so when the number of independent variables is higher. In that case, we can plot the scatter plot of **residuals vs predicted values** and check if the variance is constant or not.

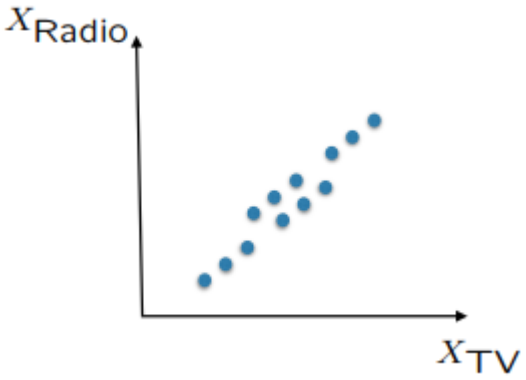
In general, **outliers** are a good example of data points that make the distribution heteroscedastic, because they create high variance in the distribution.

Multicollinearity

It is a phenomenon when there is a correlation between the independent features in a dataset. The linear regression model is not applicable in such a case. Since it assumes no correlation between the independent features, multicollinearity causes repetition of information shared to the model.

In case of multicollinearity, the matrix of independent features, namely X, is not a **full rank** matrix, and hence $X^T X$ is not invertible. Since calculation of standard errors involve $(X^T X)^{-1}$, we need to matrix X to be invertible. To make it a full rank matrix, it is required to remove some of the features. Removing features from the set of multicollinear features will make the information content unique.

Let us understand this with the help of the below figure. X_{Radio} and X_{TV} are the two independent features respectively on the Y and the X-axis. It can be seen that the two features are showing a **strong correlation** with each other. Now, using both of them in a single model will create a duplicate of information for the model. So it is good to remove either X_{Radio} or X_{TV} from the model.



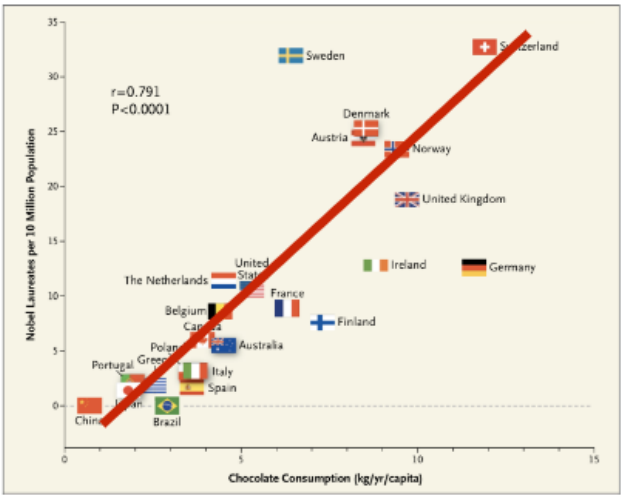
Remarks:

1. A square matrix has full rank if all the columns are **linearly independent**.
2. A set of vectors is linearly dependent if there is a nontrivial linear combination of the vectors that equals 0, otherwise the set of vectors will be linearly independent.
3. If some correlation exists between the features, then it is not a problem because it does not create the total duplicate of any of the features and the matrix X might still has full rank.

It happens in applicaton that there seems to be a relation existing between two features, the correlation will also be high, but in fact, changing one will not necessarily change the other. There are cases where making a change in one feature will not nudge the other in either way. Let us understand this below.

Prediction versus modeling

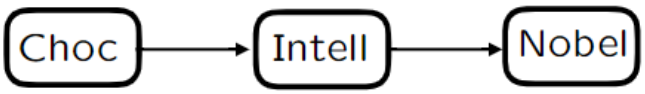
It is possible that there is a correlation between two features that are not sensibly relatable in real life. For example, there is a strong relationship found between the amount of chocolate consumed and the number of Nobel prizes won by different countries.



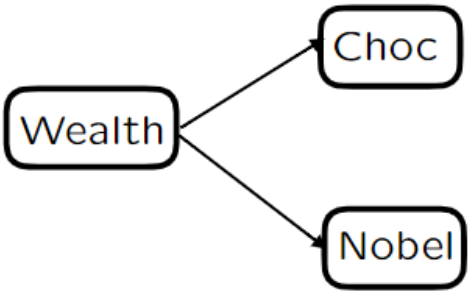
The NEW ENGLAND JOURNAL of MEDICINE

Below are some of the possible theories created out of that:

a. Chocolate makes people intelligent and hence they win more Nobels. In this case, there is a **causal relationship** between consumption of chocolate and the number of Nobel prizes.



b. Countries with more wealth are consuming more chocolate and earning more Nobels. In this case, there is no causal relationship between chocolate consumption and the number of Nobels. They are correlated due to their dependence on a third factor, wealth. This shows that **correlation does not imply causation**.



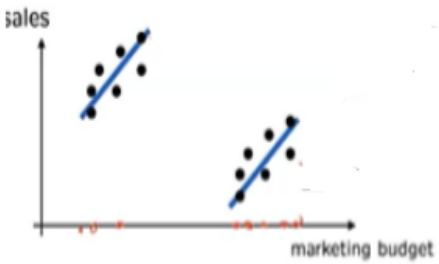
One can use the above relation to make predictions but that cannot be theoretically supported. We cannot model the relationship, or in other words, we cannot extract meaningful insights from the model.

In the above example, wealth is considered to be a **latent variable**. The latent variables are variables that are not directly observed but are rather inferred through a mathematical model from other variables that are observed.

When there are such latent variables, the independent features become correlated with residuals. This phenomenon is commonly known as **endogeneity**.

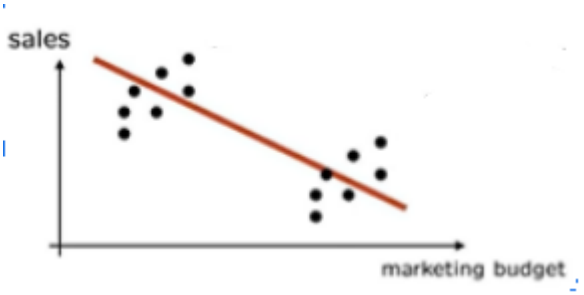
Modeling is hard in the presence of latent variables

Let's consider a scenario. Advertisements are done as an experiment in two different parts of a town. In the first part, there is not much competition, and the sales are good even with a little advertisement. In the second part, there are competing stores/dealers, and needs more advertisement for more sales, although not as good as the first part. The following figure shows the regression lines for both parts of the town.



In both the towns, the regression line shows that as the marketing budget is increasing, sales are also increasing. So, it can be concluded that increasing the marketing budget will increase sales.

Now, let us fit the regression line on both parts of the town. It shows that the sales are decreasing with the increase in the marketing budget. That is not true and it can be deceiving. Such an effect is due to the latent variable, competition.



This is called **Simpson's paradox**, in which a trend appears in several groups of data but disappears or reverses when the groups are combined.

In such cases, we can still make good predictions, but we do not get the right structural model, we cannot answer “what if” questions, and the formulas for standard errors, etc., do not hold. It is a big issue in social and bio/medical science.

Mitigation: use more variables

The issue of latent variables can be solved by adding extra variables to the model. For example, if market size is important to the model, then add market size to it. In terms of adding variables to the model, one of the problems is what variable to add? It is not possible all the time to collect all the relevant data.

The individual features in linear regression are generally linear features. It has been observed that adding non-linear features (created by either transformation of one of the existing features or the combination of two or more features) to the model is equivalent to adding new variables to the model. Using such features in the model is called Linear regression with non-linear features, which is explained below.

Some common transformations are taking the logarithm of a feature, square root, or some other functional mapping, based on how what is more relevant to the problem. In a combination of features, products of two or more features, or some other nonlinear combination of features can be taken into account.

If the variable is transformed but is now useful for the model, it is still a linear regression problem but with more features. The nonlinear features might be a combination of more than two inherent features but mathematically it will still be a least square problem. So adding nonlinear features still keeps the mathematics behind it a linear equation.

Note: The algorithm is still called lienar regression after adding nonlinear features because the linearity assumption in linear regression means the model is linear in parameters (i.e. coefficients of variables) and may or may not be linear in variables.

It is also feasible that when the combination of two features is useful but not the individual features. In that case, we can replace the original features with with the new combined feature. In case we don’t know, it is better to keep old as well as new features.

Let us have a look at the equation of the model we got for our advertising example:

$$\widehat{Sales} = 2.94 + 0.046 \times (TV) + 0.19 \times (Radio) - 0.001 \times (News)$$

Here, $R^2 = 0.897$

This is a simple model where no transformations or combinations of features are used.

Now, let us add a feature to the model which is the product of two existing features, TV and Radio.

$$\widehat{Sales} = 2.94 + 0.046 \times (TV) + 0.19 \times (Radio) + 0.001 \times (TV).(Radio)$$

Here, $R^2 = 0.968$

It can be observed that by adding new feature, the model is giving a better R^2. So, it is advisable in such a case to retain the product of the features along with the original features.

Adding too many non-significant variables to the model may lead to overfitting in the model. The model will start capturing the noise of the data, and will not be able to make relevant, generic predictions over unseen data.

Overfitting

Adding more and more variables may seem good while assessing the performance of the model on the training data but adding an excessive number of features will lead to overfitting in the model. On adding more features, it will start following the noise. This will lead to problems in predicting new or unseen data. So, we need to have a way to decide when to stop the addition of further variables.

One of the foremost methods to deal with overfitting is regularization. Two most common regularization techniques that are found useful to deal with overfitting in linear regression models are **Ridge** regularization and **Lasso** regularization. Let us understand them one by one.

Regularization: Ridge Regression

In Ridge Regression, the loss function consists of an extra term called the regularization term. The parameters are not allowed to follow the noise too much. They will be kept in control by penalizing them. This may be understood as trying to fit the noise in an economical way. If the penalization factor α is 0, then no regularization is done. If it is greater than zero, then it is a regularized regression. Mathematically, the loss function of ridge regression can be given as follows:

$$Regularized\ loss\ function = loss\ function + \alpha \sum_{j=1}^{n} \theta_j^2$$

Where, α is the **regularization hyperparameter**.

Along with Ridge regression, there is another way of doing the regularization that is named Lasso regression. The only difference is there in the **regularization** term in the loss function. Let us see it below in a bit more detail.

Regularization: Lasso Regression

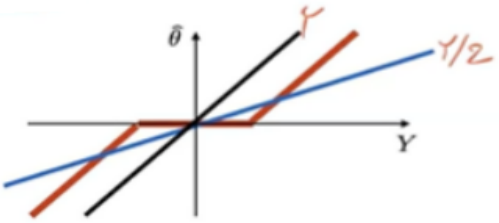
Here, the modulus of the weights is used as an additive term to the loss function of linear regression. It sets many of the weights to zero and this way it will make the model sparse. When α is big, weights are small. Larger α means more sparse solutions. Mathematically, it can be given as follows:

$$Regularized\ loss\ function\ =\ loss\ function\ +\ \alpha \sum_{j=1}^{n} |\theta_j|$$

Sparsity means if X vector is of dimension 100, θ will also be of dimension 100. Out of this 100, if θ is 0 most of the time then it is a sparse matrix.

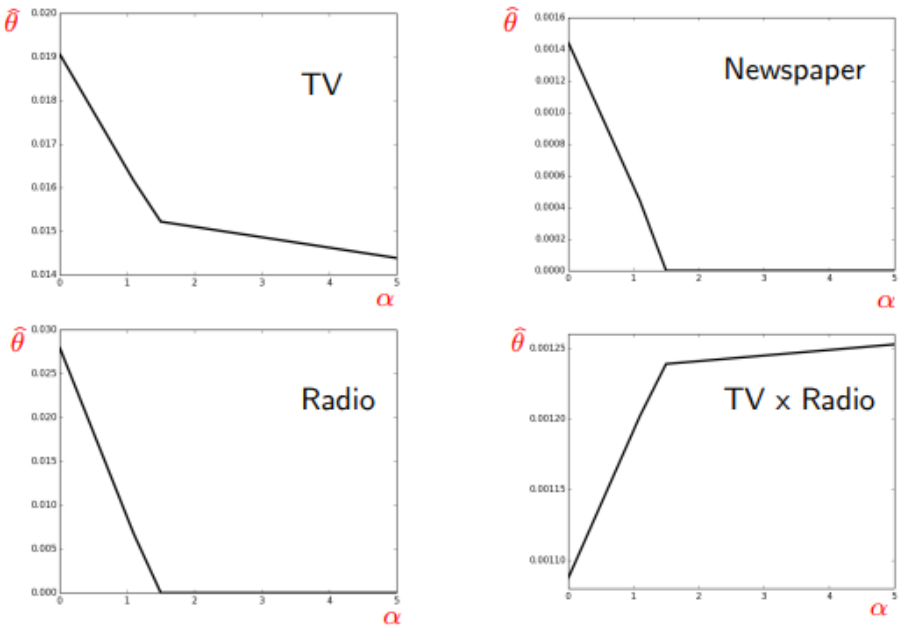
In general, to find the best working regularization methods, it is advisable to try both of them with different values of the regularizartion hyperparameter and use the one working the best.

The below diagram is drawn between the weights and the output feature. It can be understood that the Ridge regression has a bias in favor of zero. It moves the weights towards zero. With Lasso, if the weight is small then bring it to zero. The black line is the regression line without regularization. The blue line is Ridge regression while the red line is the Lasso regression. In the Lasso regression, it can be observed that when the weights are close to zero, it is suppressing them to zero. So, Lasso penalizes small weights to totally zero.



Marketing example: Lasso

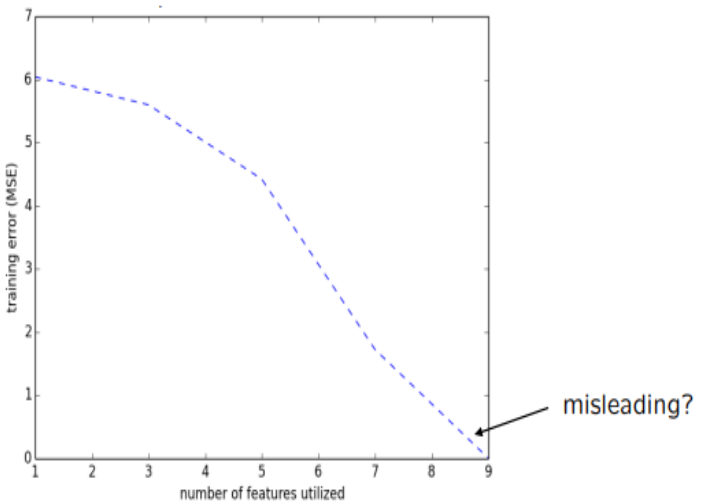
As we know in Lasso regression, if the weights are getting closer to zero, they will be made exactly zero. From the below figure, there is a variation of α with θ, with θ on the y-axis and α on the x-axis. It can be observed that for newspaper and radio θ is made exactly zero as α increases, hence these two variables will vanish because their weights are made 0. The other two features that are tv and the combination of tv and radio are retained.



In the abundance of features, it happens many times that there are too many optimal predictors to choose from. In such a case, selecting good features becomes a challenging task. To resolve such a case, the first step is to fit the model on the existing data, and then make predictions on the unseen data from the same population. This is part of the generalization of the model. Then the model that is generalizing better has to be selected as the final model with a certain set of features as input.

It has been a common observation that when we increase the number of features in the model, the training error drops. This happens because as the number of features increases, it starts capturing the noises of the data itself. Let us go through the below figure.

Here, the y-axis represents the training error and the x-axis depicts the number of features utilized. As the number of features increases, the training error goes down.



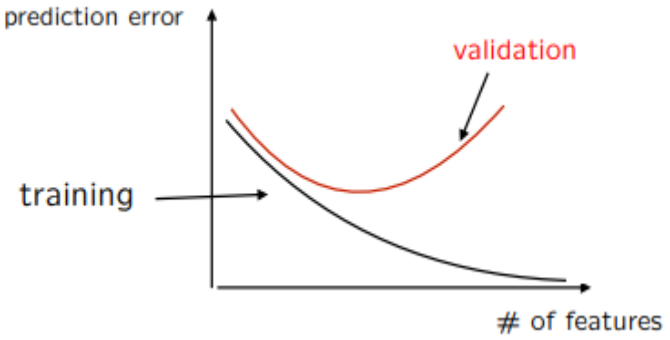
The model seems to fit better if we increase the number of features, but in reality, we are overfitting the model on the training data. Overall, we need a model that is more generic on new data. To do so, let us understand how to manage the training error (bias) and the testing error (variance) using the concept of **Bias-variance tradeoff**.

Bias-variance tradeoff

Loosely, bias can be interpreted as the **training error** of the model, while variance is the **testing error** of the model. In real life, we do not prefer a model with a **high bias** or a **high variance**. It must be a model with **low bias and low variance** because that model is supposed to make reliable predictions. It is found that having very few features will lead to underfitting and too many features will lead to overfitting of the model.

To resolve this, let us understand the concept of the **validation set**. A validation set is a subset of the original data that is used to validate and tune the performance of the model. It is initiated by randomly dividing the data into training and validation sets. The model is trained on the training set and validated on the validation set by making predictions on that. The validation set should be from the same population.

From the below figure, it can be observed that as the number of features increases, the training error goes down, while the validation error has a different pattern. It initially goes down but after a while, it starts going up.



The appropriate number of features belong to a point where the validation error becomes minimum, i.e., in the valley of the convex part of the figure. The validation error is high for both, a lower number of features as well as a higher number of features. This is because when the number of features are less, the model is so **simple** that it **does not** capture the patterns in the data. It learned less than enough to perform better on unseen data. However, when the number of features is too high, then the model has become too **complex**, it cannot make generic predictions on unseen data. Somewhere in the middle of these two extremes, there is a sweet spot where the model is flexible enough to learn from the training set in such a way that it uses that learning efficiently on the unseen data to make generic predictions on unseen data.

There are a few drawbacks to this validation process. They are listed below:

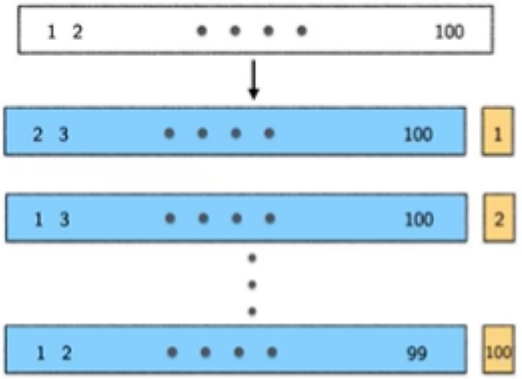
1. Some data is **wasted** and not used for training

2. Error on the **validation set** has high randomness

To resolve this, the next idea is the concept of **LOOCV** (Leave-One-Out Cross-Validation). This is explained below:

Leave-One-Out Cross-Validation

Here, the model is trained on **n-1** (n is the number of records) data points and **1** data point is kept for testing. That is why it is called Leave one out cross-validation. The process is repeated n times and will lead to making n different models. At the end, it calculates the mean of errors over the n repetitions. The process can be understood with the following figure:



In the above figure, there are 100 data points. Out of them, 99 are selected for training and hundred such models are trained.

There are certain **advantages** of this model that are listed below:

1. **No variability** due to random choice of validation set. Here, the validation set is not chosen randomly. Every data point has to be present once in the validation set.

2. **Uses full data for training**. It uses all the data points for training as only one point is left for validation at each iteration.

There are also **disadvantages** associated with this method that are as follows:

1. **Have to train n times**. As only one point is left in the validation set, there becomes a total of n iterations of training the model. This is computationally challenging and time consuming.

2. **N prediction errors are highly dependent.** As different models are having many records common in the data they are trained on, the prediction errors are highly dependent on each other.

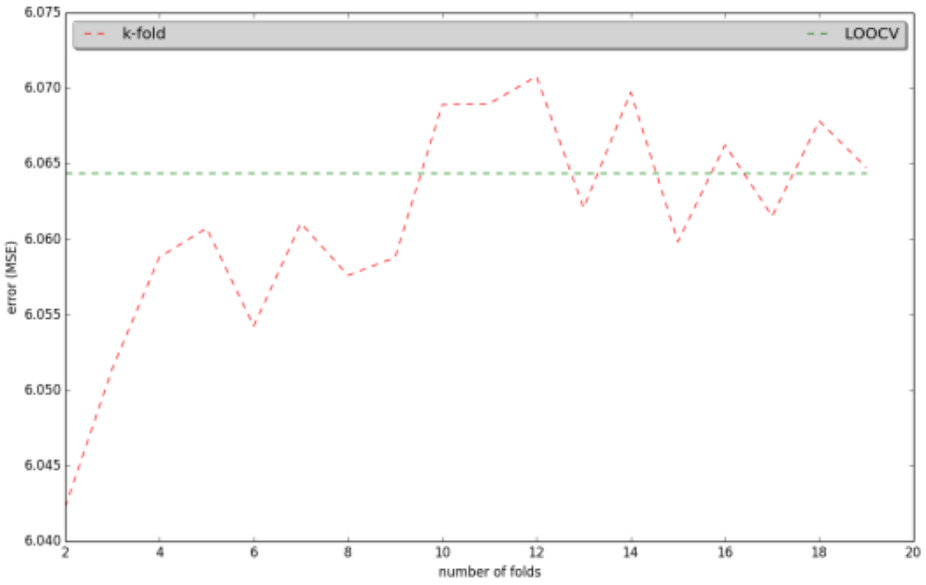
To overcome this, the concept of K-Fold Cross-Validation is introduced.

K-fold cross-validation

In this method, the entire data is divided into **K** folds. Out of them, one fold is kept for testing, while others are used for training the model. This can be understood as a more practical version of LOOCV. The steps are as follows:

1. Randomly divide the data into **K** groups.
2. For every value of K, keep one partition as a **hold-out** set and use the rest of the data for training.
3. Repeat the process K times.
4. Take the average of the error from K folds at the end.

This method can be used to find a different set of parameters and features. From the below figure, the variation of the mean squared error with the number of folds can be observed. As the number of folds increases, the error also increases. Generally, K corresponding to a certain threshold error is taken as the final K.



Commonly, it is observed that for different models trained on different samples from the same population, the error is also different. So, there is inherent variance in the performance of the model. To overcome this, one of the solutions is **bootstrapping** the model.

Data-driven bootstrap: The idea

In **bootstrapping,** from a given dataset, a large number of datasets are created using **random sampling with replacement**. In sampling with replacement, one data point is taken from the dataset and then placed back into the dataset. Again, another data point is taken at random from

the same dataset. This process runs until we have the desired number of data points in each sample. These datasets may have **duplicate** records as well because the points are sampled and then replaced back in the main dataset.

Bootstrapping is done when we can't get more data from the same population. From the same data, more data sets are created by repetition. It removes the dependency of output from a **single data point**. This way it handles the variance in the data.

We can find the coefficients for each sample and plot them using a histogram. This distribution is called a sampling distribution as we can plotting some estimate from multiple samples of the original data.

We can have more confidence in the estimates reporting by the sampling distribution due to the central limit theorem. The bootstrap method creates a new synthetic dataset of the same size as the original dataset (if size of each sample is the same as n). It creates samples by sampling from the original dataset but with replacement.

We think $\hat{\theta}$ in terms of another estimate produced from a data sample drawn from that distribution and we repeat this process a certain number of times. Let's say we have m bootstrap samples, for which we get m number of estimates and those estimates are generated from different datasets but all of these datasets were coming from the representative distribution.

And, we calculate the average of different estimates we got and then we look at the distance of a typical estimate from the average estimate and this reflects the inherence variance in our estimates. By using that, we can calculate the standard error of the estimator.

$$\hat{\theta}ave = \frac{1}{m} \sum_{i=1}^{m} \hat{\theta}i$$

$$\widehat{Var}(\hat{\theta}) = \frac{1}{m} \Sigma_{i=1}^{m} (\hat{\theta}i - \hat{\theta}ave)^2$$

$$\widehat{se}(\hat{\theta}) = \sqrt{\widehat{Var}(\hat{\theta})}$$

The general idea is to learn something about the distribution of the estimates by resampling, by reusing the data in a clever way many times.