Model Evaluation: Cross-Validation & Bootstrapping

In supervised learning regression problems, **linear regression** has been one of the dominant algorithms to give an appropriate model for relevant predictions. As it is a simple algorithm to use and make predictions, it is also one of the preferred algorithms to tackle regression problems. Similar to other algorithms linear regression has some assumptions upon which it acts. Let us understand the assumptions of linear regression first in-depth.

Assumptions of linear regression -

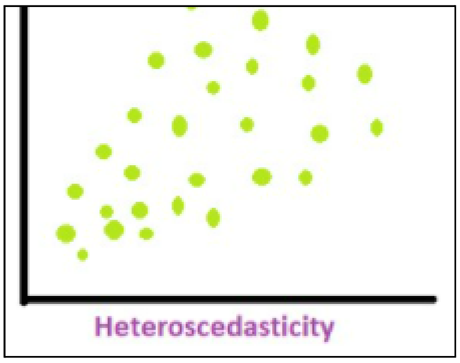
Below are the assumptions of linear regression given in detail -

1. **Heteroscedasticity** - To get a better fit of the model in a linear regression algorithm it is required to have a **constant** variance of **residuals** along the regression line/best-fit line. It makes the model give more generic predictions. If the variance of residuals along the line of regression is not constant, this case is called **Heteroscedasticity**. Linear regression assumes that the variance along the regression line is **constant** and the model is not heteroscedastic.

   The general equation of hypothesis in linear regression is as follows -

   $$y = a_1x_1 + a_2x_2 + ... + a_nx_n + w$$

   Where $a_1, a_2, ..., a_n$ are the weight terms.



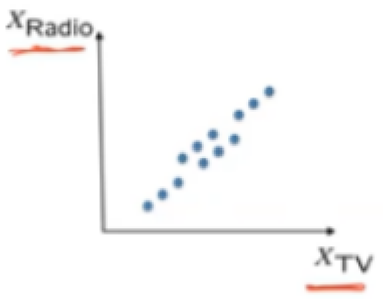
   In the above figure, it can be seen that as x is increasing the **residuals** are also increasing or it is not constant. In such a case linear regression is not supposed to function efficiently. This is a case of **heteroscedasticity**. In general, **outliers** are a good example of data points that make the distribution heteroscedastic, because they create high variance in the distribution.

   To resolve such an issue one can use the **weighted least squares** criterion, where **less weightage** is given to **high variance** points and **high weightage** to **low variance** points.

2. **Multicollinearity** - Multicollinearity is a phenomenon where there exists a **correlation** between the **independent features** in a dataset. The linear regression model is not applicable in such a case, since it assumes beforehand that there is no **correlation** between the independent features. Multicollinearity causes **repetition** of information shared by the model.

   Multicollinearity has an effect on the model. The matrix of independent features namely X is not of **full rank** matrix and hence it is not **invertible**. To make it a full rank matrix it is needed 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 below figure. $X_{Radio}$and $X_{TV}$are the two independent features respectively on the y-axis and x-axis. It can be seen that the two features are showing **perfect 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.



If multicollinearity exists for a **segment** of data then it is **not** a problem. This is because it does not create the total duplicate of any of the features.

It happens in real-life cases 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 -

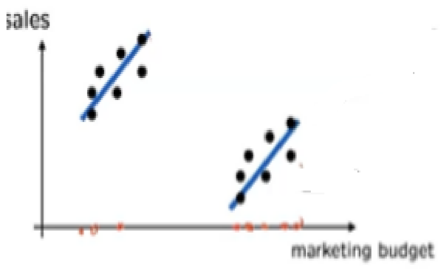
It is possible that there is a **correlation** between two features that are not sensibly relatable in real life. For example, there is a **strong relationship** found between the amount of **chocolate** consumed and the **number of Nobel prizes** won by different countries. One can make predictions based on that but that can not be theoretically supported. Below are some of the possible theories created out of that -

a. **Chocolate** makes people intelligent and hence they **win** more Nobels.
b. Countries with **high wealth** are consuming **more chocolate** and hence they earn more Nobels.

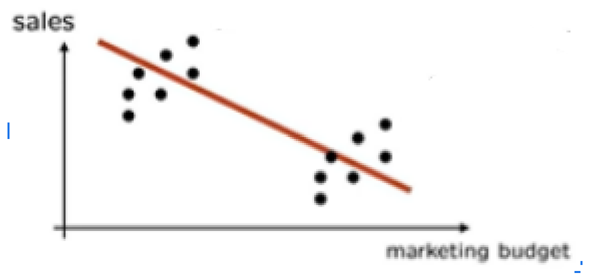
As above we understood the correlation between one independent variable with the other, namely **multicollinearity**. One of the other possible scenarios is there is a correlation between an independent feature and the error term. Such a case is called endogeneity.

Endogeneity

To understand endogeneity let us take an example. Different sorts of advertisements are done as an experiment in **different towns** and it is seen that the **marketing budget** is affecting the **sales** directly. Let us have a look at the following diagram. In the first town if the budget is increasing, sales are also increasing and so is the case with the second town.



From here it can be concluded that increasing the marketing budget will increase sales. Now without running the experiment let us run linear regression. It can be found that the marketing budget is **decreasing** with sales. It can be seen in the below figure.



That is not true and it can be deceiving. Such an effect is called **endogeneity**. The regression may seem to give good results but nothing is learned in actuality.

Endogeneity is the case where one of the independent variables is correlated with the **error** term of the model. Let us understand this with an example. Suppose x and y are the two products whose marketing is done. Advertising x is determined by market size z and the sales y

is also determined by z. $x$can be used to predict y as per the scenario of high correlation between them. According to this, sales are dependent on the budget of advertisement. It means increasing the advertisement budget must increase the sales. But increasing the **advertisement budget** does not actually enhance sales. Data does not really tell us the effect of advertisement on sales. It means that the predictions are good but predictions of possible effects are wrong. It is a big issue in social and bio/medical science.

To resolve such cases it needs something called a **controlled experiment**. For example, out of 10 markets, 5 of them do a certain kind of advertisement and see the effect on sales while the other 5 do it with a different advertisement, and then see if there really is an effect. If there is, then the relationship can be taken as actually existing, otherwise, it will be called a data-driven one.

Apart from controlled experiments, there are a few other ways of resolving endogeneity. Let us have a look below -

Mitigating endogeneity: use more variables.

Endogeneity can be solved by adding extra variables to the model. If market size is important to the model then add market size to the model. 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 some more **relevant** data and beyond that finding a good relevant feature that is really suitable to the scenario.
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 more than one feature) 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 -

One **transformational** example is taking the **logarithm** of a feature. **Square root** or any functional mapping might be a good transformation based on how relevant it is to the problem. In a **combination** of features, **products** of two or more features or **addition** or some other **linear combination** of features can be taken into account. Doing so does not come out of linear regression. It is just an augmented input feature.

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.

It is also feasible that when the **combination** of two features is useful but not the **original** feature then the original can be replaced with the combination. But if it is not sure then it is better to keep both of them.

Let us have a look at the following equations.

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

Here $R^2 = 0.897$

This is a simple model where no transformation and combination are used in features to make changes.

Now let us add a feature that is a combination of two existing features in the model.

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

Here $R^2 = 0.968$

It can be observed that by adding the product of $(TV)$and $(Radio)$it 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.

It is also needed to take care of **standard error** after **adding** such variables. There are two possibilities in terms of R^2 also. One is **simple R-squared** and the other is **adjusted R-squared**. If the amount of data is **very small** there may be a difference between them but if the amount of data is **large** it does not create any difference.

Adding more and more variables to the model may lead to overfitting the model. The model will start **capturing the noise** of the data and hence it will become difficult for it to make **relevant, generic** predictions over unseen data. So let us understand it and see how it can be resolved -

Overfitting

Adding more and more variables may seem good from the accuracy perspective but adding an excessive number of features will lead to overfitting of the model. On adding more features it will start following the noise. This will lead to problems in predicting new data 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 the **regularization** of the model. There are basically two types of regularization that are found useful to deal with the overfitting of linear regression models. They are namely **Ridge** regression and **Lasso** regression. Let us understand them one by one.

Regularization: Ridge regression

In ridge regression, the **cost/loss** function consists of an extra term called a **regularization term**. The parameters are not allowed to follow the noise **too** much. They will be kept in control by **penalizing** them in regularisations. This may be understood as trying to fit the **noise** in an **economical** way. If the penalization factor α is 0 It is **normal** linear regression. But if it is **not** zero then it is a **regularised** regression. It is a bit of a difficult mathematical problem but not more complex. Ridge regression does not allow the parameters to **change a lot**. So they are weighted. Mathematically the loss function can be given as follows -

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

Where α is called a **regularization hyperparameter**.

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

Regularization: Lasso (Sparsity enforcing)

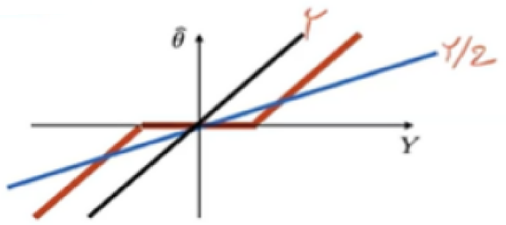
Here the **modulus** of the weights is used as an **additive** to the loss function. It sets many of the weights to **zero** and this way it will make the model **sparse**. When α is big, weight is small. Larger α means more sparse solutions. Mathematically it can be given as follows:-

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

Sparsity means if x vector is of dimension 100 so 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.

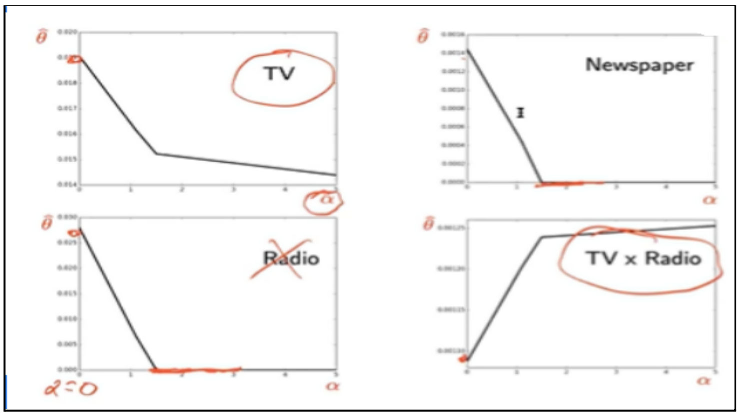
In general to find the best working regularisation methods it is advisable to try both of them and use the one working better than the other.

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



Marketing example: - Lasso

As we know in lasso regression, if the parameter or weights are getting closer to zero they will be made exactly zero by penalizing. From the below figure that is a variation of α with θ, with θon the y-axis and α on the x-axis can be seen that as for newspaper and radio as α goes closer to zero it is made exactly zero and 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.

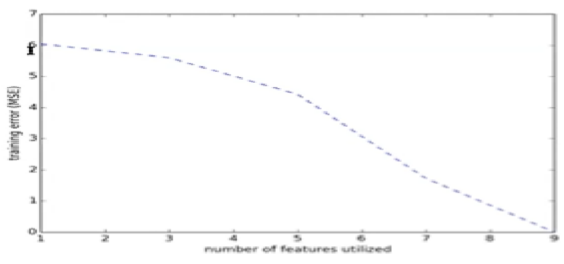


As we can see here if α increases two of the variables become zero and they are already removed. Only the remaining features are kept and the model will continue with them only. Newspapers and radio are removed. While tv and the combination are kept to continue ahead.

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. Then make predictions on the unseen/out of sample 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 whatever 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 accordingly. 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.



on the data set used to train the model...

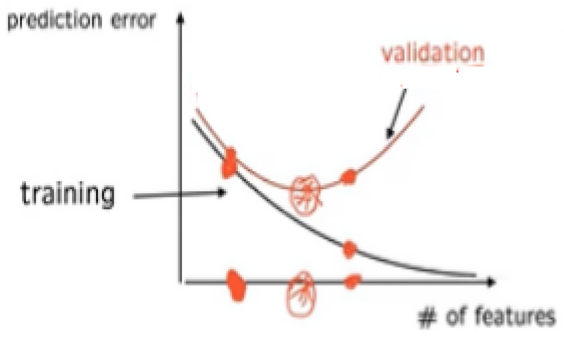
The model seems to fit better if we increase the number of features. But in such a case, we are overfitting the model on the data/noise. Overall we need a model that is more generic on new data. To do so let us understand how to manage the training error (Bias) and the testing error (variance) using what is called the Bias-variance tradeoff.

Bias-variance tradeoff

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 **high bias or high variance**. It must be a model with **low bias and low variance** because that is the model that is supposed to make reliable predictions. It is found that too few features will lead to underfitting and too many features will lead to overfitting (High variance will be there) of the model.

To resolve this let us understand the concept of the **validation set**. A validation set is a set of data that is used to validate 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 of 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 so **complex** that it can not 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.



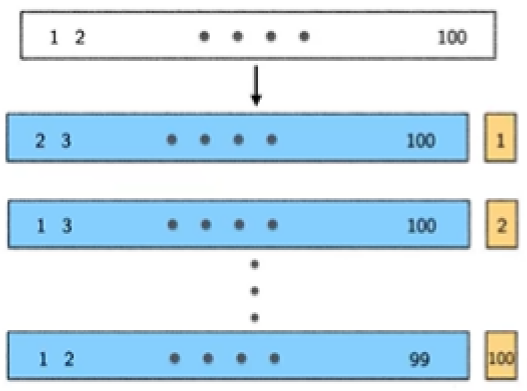
There are a few drawbacks to this validation process. They are listed below -

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 as follows -

Leave one out cross-validation

Here the model is trained on **n-1** 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 as it will lead to making n different models. Then at the end calculate the **mean squared error** over the n repetitions. The one with the **least error** is taken as the best model. 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 a 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 all data for training** -
   It uses almost all the data points for training as only one point is left for validation.

There are certain **disadvantages** also 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 a bit challenging and time taking too.

2. **N** predictions errors are highly dependent.

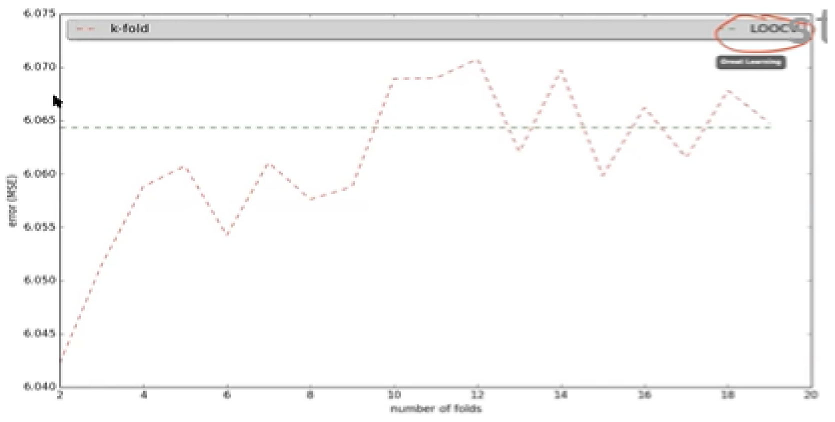
To overcome this the final concept is of K-fold cross-validation. That is detailed as below -

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. It can be understood as follows -

1. Randomly divide the data into **k** groups.
2. For every value of k, keep it as a **hold-out** set and use the rest of the data for the training set.
3. Keep on recording the **mean squared error** and take the average of this at the end of k training

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 a number of folds can be understood. As the number of folds increases, the error also increases. Generally, **k** corresponding to a certain threshold error is taken as final.



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 put 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 sets. These datasets may have **duplicate** records as well, since the points are sampled and then replaced back in the main dataset.

Bootstrapping is done when we don't have the choice of selecting more data sets from the 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.