Chapter 3

Common Issues in Machine Learning

I. Datasets

Training, Validation and Test Datasets

It is a good practice to use a separate dataset to test the performance of your algorithm, as testing the algorithm on the training set may not give you the true generalization power of the algorithm. In most real world use cases, based on the validation accuracy, we often tweak our algorithm in different ways, such as adding more layers or different layers, or using different techniques. So, there is a higher chance that your choices for tweaking the algorithm are based on the validation dataset. Algorithms trained this way tend to perform well in the training dataset and the validation dataset, but fail to generalize well on unseen data. This is due to an information leak from your validation dataset, which influences us in tweaking the algorithm.

To avoid the problem of an information leak and improve generalization, it is often a common practice to split the dataset into three different parts, namely a training, validation, and test dataset. We do the training and do all the hyper parameter tuning of the algorithm using the training and validation set. At the end, when the entire training is done, then you will test the algorithm on the test dataset. There are two types of parameters that we talk about. One is the parameters or weights that are used inside an algorithm, which are tuned by the optimizer or during backpropagation. The other set of parameters, called hyper parameters, controls the number of layers used in the network, learning rate, and other types of parameter that generally change the architecture, which is often done manually. The phenomenon of a particular algorithm performing better in the training set and failing to perform on the validation or test set is called overfitting, or the lack of the algorithm's ability to generalize. There is an opposite phenomenon where the algorithm fails to perform for the training set, which is called underfitting. We will look at different strategies that will help us in overcoming the overfitting and underfitting problems. Let's look at the various strategies available for splitting the dataset before looking at overfitting and underfitting.

Training, Validation, And Test Splits

It is best practice to split the data into three parts. Training, validation, and test datasets. The best approach for using the holdout dataset is to:

1. Train the algorithm on the training dataset

2. Perform hyper parameter tuning based on the validation dataset

3. Perform the first two steps iteratively until the expected performance is achieved

4. After freezing the algorithm and the hyper parameters, evaluate it on the test dataset. Avoid splitting the data into two parts, as it may lead to an information leak. Training and testing it on the same dataset is a clear no-no as it does not guarantee algorithm generalization. There are three popular holdout strategies that can be used to split the data into training and validation sets. They are as follows:

- Simple holdout Validation
- K-fold Validation
- Iterated k-fold Validation

Simple holdout validation set apart a fraction of the data as your test dataset. What fraction to keep may be very problem-specific and could largely depend on the amount of data available. For problems particularly in the fields of computer vision and Natural Processing Language (NLP), collecting labeled data could be very expensive, so to hold out a large fraction of 30% may make it difficult for the algorithm to learn, as it will have less data to train on. So, depending on the data availability, choose the fraction of it wisely. Once the test data is split, keep it apart until you freeze the algorithm and its hyper parameters. For choosing the best hyper parameters for the problem, choose a separate validation dataset. To avoid overfitting, we generally divide available data into three different sets, as shown in the following table:

Available Datasets		
Training Set	Validation Set	Test Set

We used a simple implementation to create our validation set. This is one of the simplest holdout strategies and is commonly used to start with. There is a disadvantage of using this with small datasets. The validation dataset or test dataset may not be statistically representative of the data at hand. We can easily recognize this by shuffling the data before holding out. If the results obtained are not consistent, then we need to use a better approach. To avoid this issue, we often end up using k-fold or iterated k-fold validation.

K-fold validation Keep a fraction of the dataset for the test split, then divide the entire dataset into k-folds where k can be any number, generally varying from two to ten. At any given iteration, we hold one block for validation and train the algorithm on the rest of the blocks. The final score is generally the average of all the scores obtained across the k-folds.

One key thing to note when using the k-fold validation dataset is that it is very expensive, because you run the algorithm several times on different parts of the dataset, which can turn out to be very expensive for computation-intensive algorithms particularly in areas of computer vision algorithms, where, sometimes, training an algorithm could take anywhere from minutes to days. So, use this technique wisely.

K-fold validation with shuffling

To make things complex and robust, you can shuffle the data every time you create your holdout validation dataset. It is very helpful for solving problems where a small boost in performance could have a huge business impact. If your case is to quickly build and deploy algorithms and you are OK with compromising a few percent in performance difference, then this approach may not be worth it. It all boils down to what problem you are trying to solve, and what accuracy means to you.

There are a few other things that you may need to consider when splitting up the data, such as:

- Data representativeness
- Time sensitivity
- Data redundancy

In the example of images, we classified images as either dogs or cats. Let's take a scenario where all the images are sorted and the first 60% of images are dogs and the rest are cats. If we split this dataset by choosing the first 80% as the training dataset and the rest as the validation set, then the validation dataset will not be a true representation of the dataset, as it will only contain cat images. So, in these cases, care should be taken that we have a good mix by shuffling the data before splitting or doing a stratified sampling. Stratified sampling refers to picking up data points from each category to create validation and test datasets.

Let's take the case of predicting stock prices. We have data from January to December. In this case, if we do a shuffle or stratified sampling then we end up with an information leak, as the prices could be sensitive to time. So, create the validation dataset in such a way that there is no information leak. In this case, choosing the December data as the validation dataset could make more sense. In the case of stock prices it is more complex than this, so domain-specific knowledge also comes into play when choosing the validation split. Duplicates are common in data. Care should be taken so that the data present in the training, validation, and test sets are unique. If there are duplicates, then the model may not generalize well on unseen data.

II. Data Preprocessing and Feature Engineering

Feature Selection Techniques in Machine Learning

Feature selection is a way of selecting the subset of the most relevant features from the original features set by removing the redundant, irrelevant, or noisy features.

While developing the machine learning model, only a few variables in the dataset are useful for building the model, and the rest features are either redundant or irrelevant. If

we input the dataset with all these redundant and irrelevant features, it may negatively impact and reduce the overall performance and accuracy of the model. Hence, it is very important to identify and select the most appropriate features from the data and remove the irrelevant or less important features, which is done with the help of feature selection in machine learning.

Feature selection is one of the important concepts of machine learning, which highly impacts the performance of the model. As machine learning works on the concept of "Garbage In Garbage Out", so we always need to input the most appropriate and relevant dataset to the model in order to get a better result.

So, let's first understand some basics of feature selection.

- **What is Feature Selection?**
- **Need for Feature Selection**
- **Feature Selection Methods/Techniques**

What is Feature Selection?

A feature is an attribute that has an impact on a problem or is useful for the problem, and choosing the important features for the model is known as **feature selection.** Each machine learning process depends on feature engineering, which mainly contains two processes; which are Feature Selection and Feature Extraction. Although feature selection and extraction processes may have the same objective, both are completely different from each other. The main difference between them is that feature selection is about selecting the subset of the original feature set, whereas feature extraction creates new features. Feature selection is a way of reducing the input variable for the model by using only relevant data in order to reduce overfitting in the model.

So, we can define feature Selection as:

"*It is a process of automatically or manually selecting the subset of most appropriate and relevant features to be used in model building*." Feature selection is performed by either including the important features or excluding the irrelevant features in the dataset without changing them.

Need for Feature Selection

Before implementing any technique, it is really important to understand, need for the technique and so for the Feature Selection. As we know, in machine learning, it is necessary to provide a pre-processed and good input dataset in order to get better outcomes. We collect a huge amount of data to train our model and help it to learn better. Generally, the dataset consists of noisy data, irrelevant data, and some part of useful data. Moreover, the huge amount of data also slows down the training process of the model, and with noise and irrelevant data, the model may not predict and perform well. So, it is very necessary to remove such noises and less-important data from the dataset and to do this, and feature selection techniques are used.

Selecting the best features helps the model to perform well. For example, suppose we want to create a model that automatically decides which car should be crushed for a spare part, and to do this, we have a dataset. This dataset contains a Model of the car, Year, Owner's name, Miles. So, in this dataset, the name of the owner does not contribute to the model performance as it does not

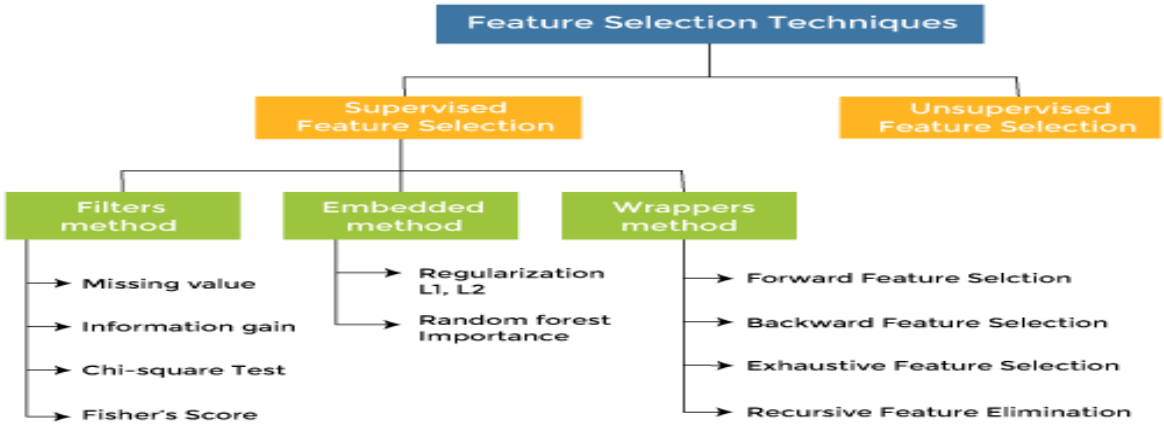
decide if the car should be crushed or not, so we can remove this column and select the rest of the features (column) for the model building.
The following are some benefits of using feature selection in machine learning:

- **It helps in avoiding the curse of dimensionality.**
- **It helps in the simplification of the model so that it can be easily interpreted by the researchers.**
- **It reduces the training time.**
- **It reduces overfitting hence enhance the generalization.**

Feature Selection Techniques

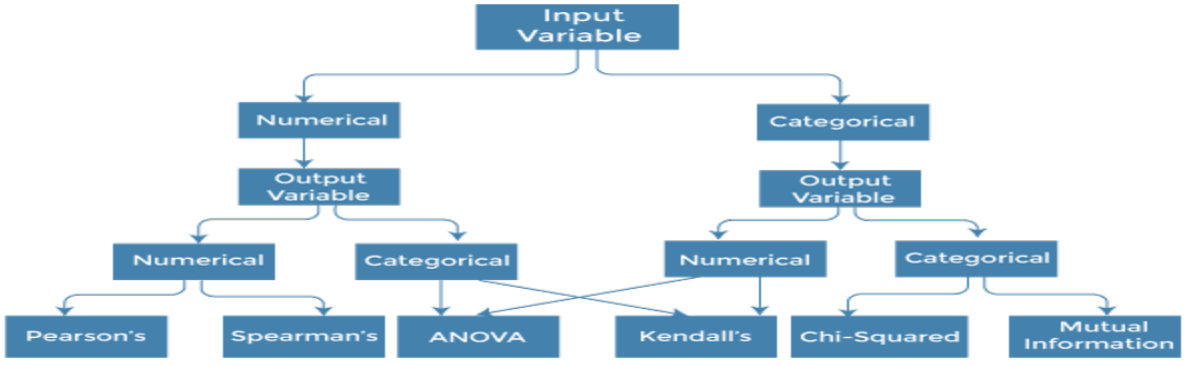
There are mainly two types of Feature Selection techniques, which are:

- **Supervised Feature Selection technique**: Supervised Feature selection techniques consider the target variable and can be used for the labelled dataset.
- **Unsupervised Feature Selection technique**: Unsupervised Feature selection techniques ignore the target variable and can be used for the unlabelled dataset.



How to choose a Feature Selection Method?

For machine learning engineers, it is very important to understand that which features selection method will work properly for their model. The more we know the datatypes of variables, the easier it is to choose the appropriate statistical measure for feature selection.



To know this, we need to first identify the type of input and output variables. In machine learning, variables are of mainly two types:

- **Numerical Variables:** Variable with continuous values such as integer, float
- **Categorical Variables:** Variables with categorical values such as Boolean, ordinal, nominals.

The following are some univariate statistical measures, which can be used for filter-based feature selection:

1. Numerical Input, Numerical Output:

Numerical Input variables are used for predictive regression modelling. The common method to be used for such a case is the Correlation coefficient.

- Pearson's correlation coefficient (For linear Correlation).
- Spearman's rank coefficient (for non-linear correlation).

2. Numerical Input, Categorical Output:

Numerical Input with categorical output is the case for classification predictive modelling problem**s.** In this case, also, correlation-based techniques should be used, but with categorical output.

- **ANOVA correlation coefficient (linear).**
- **Kendall's rank coefficient (nonlinear).**

3. Categorical Input, Numerical Output:

This is the case of regression predictive modelling with categorical input. It is a different example of a regression problem.

4. Categorical Input, Categorical Output:

This is a case of classification predictive modelling with categorical Input variables. The commonly used technique for such a case is Chi-Squared Test. We can also use Information gain in this case.

We have looked at different ways to split our datasets to build our evaluation strategy. In most cases, the data that we receive may not be in a format that can be readily used by us for training our algorithms. So, we will cover some of the preprocessing techniques and feature engineering techniques. Though most of the feature engineering techniques are domain-specific, particularly in the areas of computer vision and text, there are some common feature engineering techniques that are common across the board.

Data preprocessing for neural networks is a process in which we make the data more suitable for the deep learning algorithms to train on. The following are some of the commonly-used data preprocessing steps:

- Vectorization
- Normalization
- Missing values
- Feature extraction

Vectorization: Data comes in various formats such as text, sound, images, and video. The very first thing that needs to be done is to convert the data into tensors. For problems involving structured data, the data is already present in a vectorized format; all we need to do is convert them into tensors.

Value Normalization: It is a common practice to normalize features before passing the data to any machine learning algorithm or deep learning algorithm. It helps in training the algorithms faster and helps in achieving more performance. Normalization is the process in which you represent data belonging to a particular feature in such a way that its mean is zero and standard deviation is one.

For image data, it is also a common practice to divide each pixel value by 255 so that all the values fall in the range between zero and one, particularly when you are not using pre-trained weights. Normalization is also applied for problems involving structured data. Say we are working on a house price prediction problem. There could be different features that could fall in different scales. For example, distance to the nearest airport and the age of the house is variables or features that could be in different scales. Using them with neural networks as they are could prevent the gradients from converging. In simple words, loss may not go down as expected. So, we should be careful to apply normalization to any kind of data before training on our algorithms.

To ensure that the algorithm or model performs better, ensure that the data follows the following characteristics:

Take small values: Typically in a range between zero and one.

Same range: Ensure all the features are in the same range.

Handling missing values: Missing values are quite common in real-world machine learning problems. From our previous examples of predicting house prices, certain fields for the age of the house could be missing. It is often safe to replace the missing values with a number that may not occur otherwise. The algorithms will be able to identify the pattern. There are other techniques that are available to handle missing values that are more domain-specific.

Feature engineering: Feature engineering is the process of using domain knowledge about a particular problem to create new variables or features that can be passed to the model. To understand better, let's look at a sales prediction problem. Say we have information about promotion dates, holidays, competitor's start date, distance from competitor, and sales for a particular day. In the real world, there could be hundreds of features that may be useful in predicting the prices of stores. There could be certain information that could be important in predicting the sales. Some of the important features or derived values are:

- Days until the next promotion

- Days left before the next holiday
- Number of days the competitor's business has been open

There could be many more such features that can be extracted that come from domain knowledge. Extracting these kinds of features for any machine learning algorithm or deep learning algorithm could be quite challenging for the algorithms to perform themselves. For certain domains, particularly in the fields of computer vision and text, modern deep learning algorithms help us in getting away with feature engineering. Except for these fields, good feature engineering always helps. The problem can be solved a lot faster with less computational resource. The deep learning algorithms can learn features without manually engineering them by using huge amounts of data. So, if you are tight on data, then it is good to focus on good feature engineering.

III. Overfitting and Underfitting

In the real world, the dataset present will never be clean and perfect. It means each dataset contains impurities, noisy data, outliers, missing data, or imbalanced data. Due to these impurities, different problems occur that affect the accuracy and the performance of the model. One of such problems is Overfitting in Machine Learning. Overfitting is a problem that a model can exhibit. A machine learning model is said to be overfitted if it can't generalize well with unseen data. Before understanding overfitting, we need to know some basic terms.

Noise: Noise is meaningless or irrelevant data present in the dataset. It affects the performance of the model if it is not removed.

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.

Generalization: It shows how well a model is trained to predict unseen data.

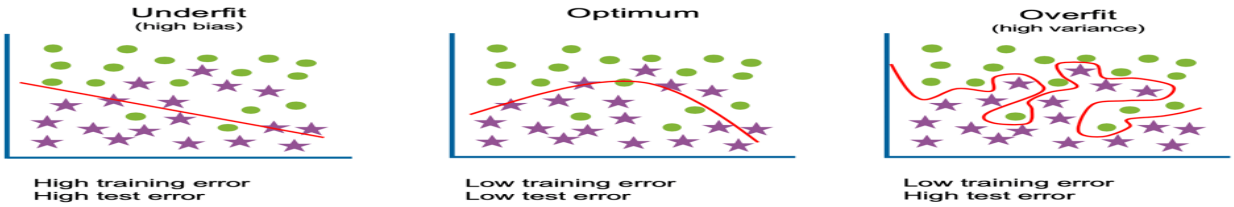
How well a model trained on the training set predicts the right output for new instances is called generalization. Generalization refers to how well the concepts learned by a machine learning model apply to specific examples not seen by the model when it was learning. The goal of a good machine learning model is to generalize well from the training data to any data from the problem domain. This allows us to make predictions in the future on data the model has never seen.

Overfitting and underfitting are the two biggest causes for poor performance of machine learning algorithms. The model should be selected having the best generalisation. This is said to be the case if these problems are avoided.

Underfitting: is the production of a machine learning model that is not complex enough to accurately capture relationships between datasets.

Overfitting: is the production of an analysis which corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably

What is Overfitting?



- Overfitting & underfitting are the two main errors/problems in the machine learning model, which cause poor performance.
- Overfitting occurs when the model fits more data than required, and it tries to capture each and every datapoint fed to it. Hence, it starts capturing noise and inaccurate data from the dataset, which degrades the performance of the model.
- An overfitted model doesn't perform accurately with the test/unseen dataset and can't generalize well.
- An overfitted model is said to have low bias and high variance.

Example to Understand Overfitting

We can understand overfitting with a general example. Suppose there are three students, X, Y, and Z, and all three are preparing for an exam. X has studied only three sections of the book and left all other sections. Y has a good memory, hence memorized the whole book. And the third student, Z, has studied and practiced all the questions. So, in the exam, X will only be able to solve the questions if the exam has questions related to section 3. Student Y will only be able to solve questions if they appear exactly the same as given in the book. Student Z will be able to solve all the exam questions in a proper way.

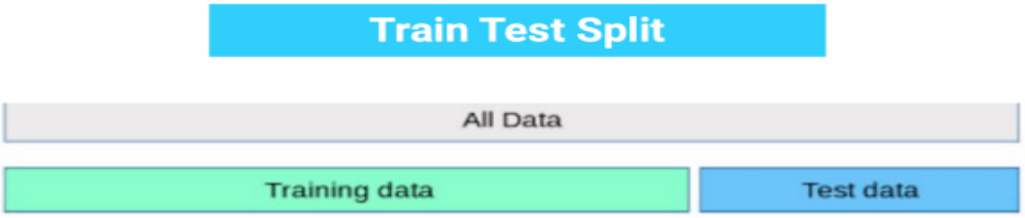
The same happens with machine learning; if the algorithm learns from a small part of the data, it is unable to capture the required data points and hence underfitted.

Suppose the model learns the training dataset, like the Y student. They perform very well on the seen dataset but perform badly on unseen data or unknown instances. In such cases, the model is said to be Overfitting. And if the model performs well with the training dataset and also with the test/unseen dataset, similar to student Z, it is said to be a good fit.

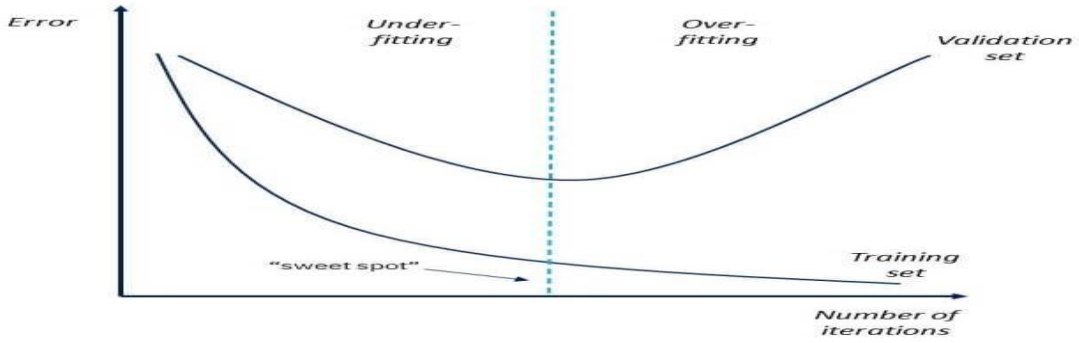
How to Detect Overfitting?

Overfitting in the model can only be detected once you test the data. To detect the issue, we can perform **Train/test** split**.**

In the train-test split of the dataset, we can divide our dataset into random test and training datasets. We train the model with a training dataset which is about 80% of the total dataset. After training the model, we test it with the test dataset, which is about 20 % of the total dataset.



Now, if the model performs well with the training dataset but not with the test dataset, then it is likely to have an overfitting issue. For example, if the model shows 85% accuracy with training data and 50% accuracy with the test dataset, it means the model is not performing well.

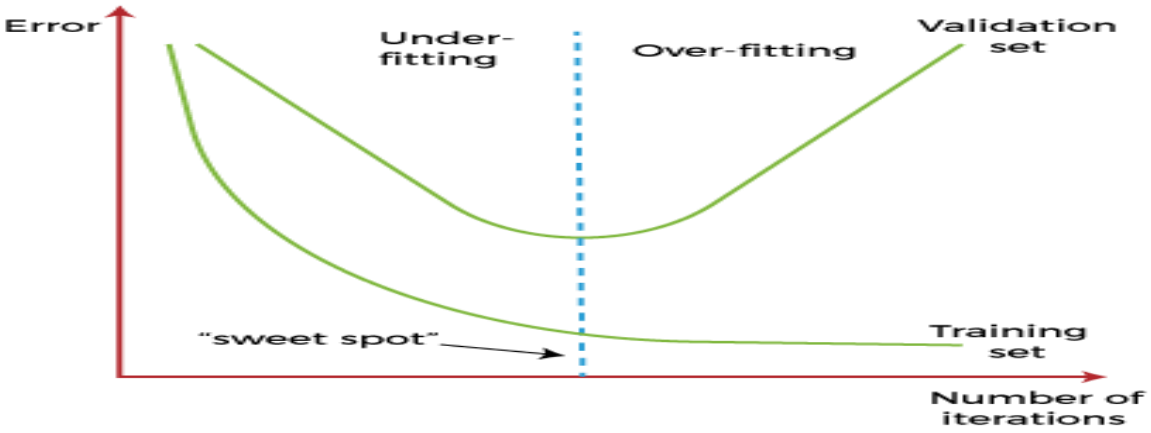


Ways to prevent Overfitting

Although overfitting is an error in Machine learning which reduces the performance of the model, however, we can prevent it in several ways. With the use of the linear model, we can avoid overfitting; however, many real-world problems are non-linear ones. It is important to prevent overfitting from the models. The following are several ways that can be used to prevent overfitting:

- **Early Stopping**
- **Train with more data**
- **Feature Selection**
- **Cross-Validation**
- **Data Augmentation**
- **Regularization**

1) Early Stopping: In this technique, the training is paused before the model starts learning the noise within the model. In this process, while training the model iteratively, measure the performance of the model after each iteration. Continue up to a certain number of iterations until a new iteration improves the performance of the model. After that point, the model begins to overfit the training data; hence we need to stop the process before the learner passes that point. Stopping the training process before the model starts capturing noise from the data is known as **Early Stopping**.



However, this technique may lead to the underfitting problem if training is paused too early. So, it is very important to find that "sweet spot" between underfitting and overfitting.

2) Train with More Data: Increasing the training set by including more data can enhance the accuracy of the model, as it provides more chances to discover the relationship between input and output variables. It may not always work to prevent overfitting, but this way helps the algorithm to detect the signal better to minimize the errors.

When a model is fed with more training data, it will be unable to overfit all the samples of data and forced to generalize well. But in some cases, the additional data may add more noise to the model; hence we need to be sure that data is clean and free from in-consistencies before feeding it to the model.

3) Feature Selection: While building the ML model, we have a number of parameters or features that are used to predict the outcome. However, sometimes some of these features are redundant or less important for the prediction, and for this feature selection process is applied. In the feature selection process, we identify the most important features within training data, and other features are removed. Further, this process helps to simplify the model and reduces noise from the data. Some algorithms have the auto-feature selection, and if not, then we can manually perform this process.

4) Cross-Validation: Cross-validation is one of the powerful techniques to prevent overfitting. In the general k-fold cross-validation technique, we divided the dataset into k-equal-sized subsets of data; these subsets are known as folds.

5) Data Augmentation: Data Augmentation is a data analysis technique, which is an alternative to adding more data to prevent overfitting. In this technique, instead of adding more training data, slightly modified copies of already existing data are added to the dataset. The data augmentation technique makes it possible to appear data sample slightly

different every time it is processed by the model. Hence, each data set appears unique to the model and prevents overfitting.

6) Regularization: If overfitting occurs when a model is complex, we can reduce the number of features. However, overfitting may also occur with a simpler model, more specifically the Linear model, and for such cases, regularization techniques are much helpful.

Regularization is the most popular technique to prevent overfitting. It is a group of methods that forces the learning algorithms to make a model simpler. Applying the regularization technique may slightly increase the bias but slightly reduces the variance. In this technique, we modify the objective function by adding the penalizing term, which has a higher value with a more complex model. The two commonly used regularization techniques are L1 Regularization and L2 Regularization.

Dropout: Dropout is one of the most commonly used and the most powerful regularization techniques used in deep learning. It was developed by Hinton and his students at the University of Toronto. Dropout is applied to intermediate layers of the model during the training time.

Underfitting:

Underfitting is just the opposite of overfitting. Whenever a machine learning model is trained with fewer amounts of data, and as a result, it provides incomplete and inaccurate data and destroys the accuracy of the machine learning model. Underfitting occurs when our model is too simple to understand the base structure of the data, just like an undersized pant. This generally happens when we have limited data into the dataset, and we try to build a linear model with non-linear data. In such scenarios, the complexity of the model destroys, and rules of the machine learning model become too easy to be applied on this data set, and the model starts doing wrong predictions as well.

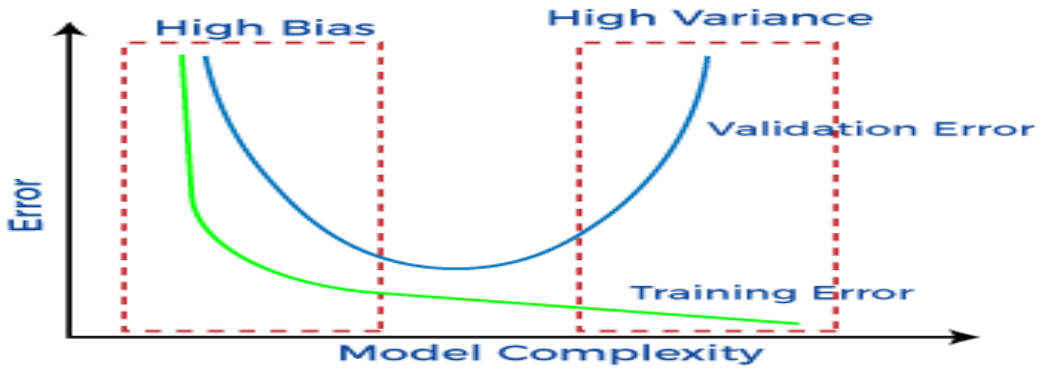
Methods to Reduce Underfitting:

- Increase model complexity
- Remove noise from the data
- Trained on increased and better features
- Reduce the constraints
- Increase the number of epochs to get better results.

IV. Bias and Variance In Machine Learning

Machine learning is a branch of Artificial Intelligence, which allows machines to perform data analysis and make predictions. However, if the machine learning model is not accurate, it can make predictions errors, and these prediction errors are usually known as Bias and Variance. In machine learning, these errors will always be present as there is always a slight difference between the model predictions and actual predictions. The main aim of ML/data science analysts

is to reduce these errors in order to get more accurate results. So, we are going to discuss bias, variance and Bias-variance trade-off. But before starting, let's first understand what errors in Machine learning are?

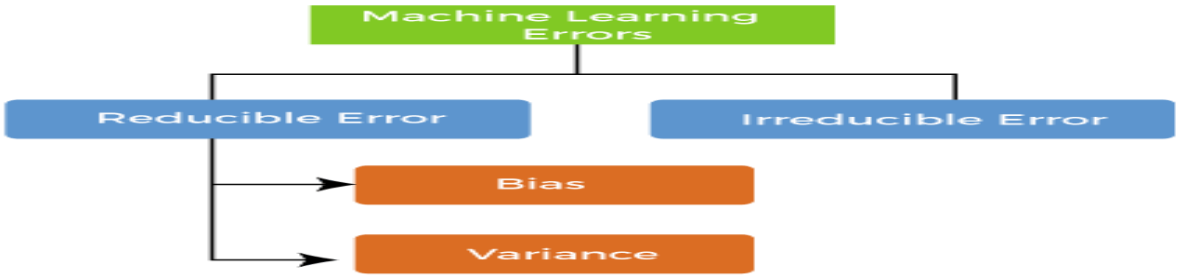


Errors in Machine Learning

In machine learning, an error is a measure of how accurately an algorithm can make predictions for the previously unknown dataset. On the basis of these errors, the machine learning model is selected that can perform best on the particular dataset. There are mainly two types of errors in machine learning, which are:

Reducible errors: These errors can be reduced to improve the model accuracy. Such errors can further be classified into bias and Variance.

Irreducible errors: These errors will always be present in the model regardless of which algorithm has been used. The cause of these errors is unknown variables whose value can't be reduced.



What is Bias?

In general, a machine learning model analyses the data, find patterns in it and make predictions. While training, the model learns these patterns in the dataset and applies them to test data for prediction. While making predictions, a difference occurs between prediction values made by the model and actual values/expected values, and this difference is known as bias errors or Errors due to bias. It can be defined as an inability of machine learning algorithms such as Linear Regression to capture the true relationship between the data points. Each algorithm begins with some amount of bias because bias occurs from assumptions in the model, which makes the target function simple to learn. A model has either:

- **Low Bias:** A low bias model will make fewer assumptions about the form of the target function.
- **High Bias:** A model with a high bias makes more assumptions, and the model becomes unable to capture the important features of our dataset. A high bias model also cannot perform well on new data.

Generally, a linear algorithm has a high bias, as it makes them learn fast. The simpler the algorithm, the higher the bias it has likely to be introduced. Whereas a nonlinear algorithm often has low bias. Some examples of machine learning algorithms with low bias are Decision Trees, k-Nearest Neighbors and Support Vector Machines**.** At the same time, an algorithm with high bias is Linear Regression, Linear Discriminant Analysis and Logistic Regression.

Ways to Reduce High Bias:

High bias mainly occurs due to a much simple model. The following are some ways to reduce the high bias:

- Increase the input features as the model is underfitted.
- Decrease the regularization term.
- Use more complex models, such as including some polynomial features.

What is a Variance Error?

The variance would specify the amount of variation in the prediction if the different training data was used. In simple words, variance tells that how much a random variable is different from its expected value. Ideally, a model should not vary too much from one training dataset to another, which means the algorithm should be good in understanding the hidden mapping between inputs and output variables. Variance errors are either of low variance or high variance.

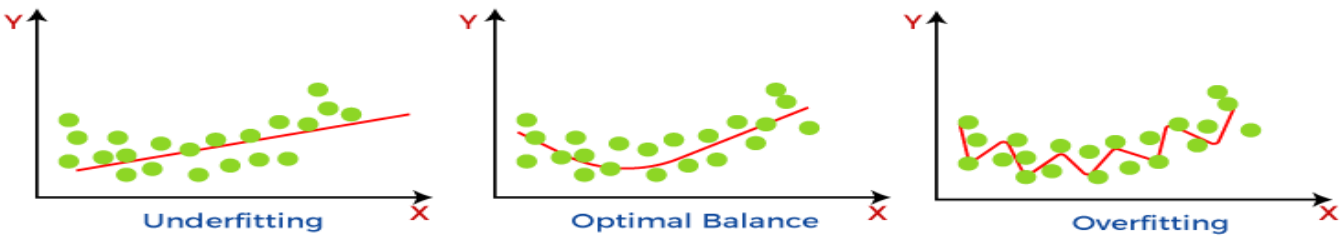
Low variance means there is a small variation in the prediction of the target function with changes in the training data set. At the same time, **High variance** shows a large variation in the prediction of the target function with changes in the training dataset.

A model that shows high variance learns a lot and perform well with the training dataset, and does not generalize well with the unseen dataset. As a result, such a model gives good results with the training dataset but shows high error rates on the test dataset.

Since, with high variance the model learns too much from the dataset, it leads to overfitting of the model. A model with high variance has the following problems.

- A high variance model leads to overfitting.
- Increase model complexities.

Usually, nonlinear algorithms have a lot of flexibility to fit the model, have high variance.



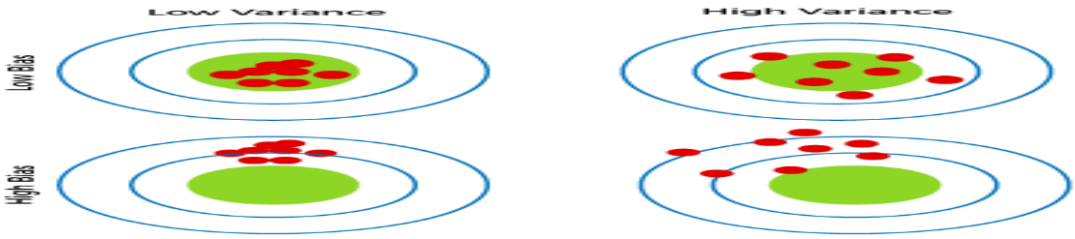
Some examples of machine learning algorithms with low variance are Linear Regression, Logistic Regression, and Linear discriminant analysis**.** At the same time examples of algorithms with high variance are Decision Tree, Support Vector Machine, and K-nearest neighbors.

Ways to Reduce High Variance:

- Reduce the input features or number of parameters as a model is overfitted.
- Do not use a much complex model.
- Increase the training data.
- Increase the regularization term.

Different Combinations of Bias-Variance

There are four possible combinations of bias and variances, which are represented by the following diagram:

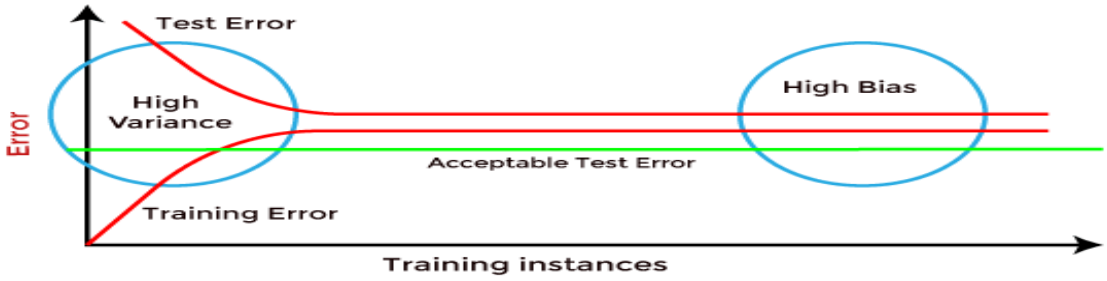


1. **Low-Bias, Low-Variance:** The combination of low bias and low variance shows an ideal machine learning model. However, it is not possible practically.
2. **Low-Bias, High-Variance:** With low bias and high variance, model predictions are inconsistent and accurate on average. This case occurs when the model learns with a large number of parameters and hence leads to an **overfitting.**
3. **High-Bias, Low-Variance:** With High bias and low variance, predictions are consistent but inaccurate on average. This case occurs when a model does not learn well with the training dataset or uses few numbers of the parameter. It leads to **underfitting** problems in the model.
4. **High-Bias, High-Variance:** With high bias and high variance, predictions are inconsistent and also inaccurate on average.

How to Identify High variance or High Bias?

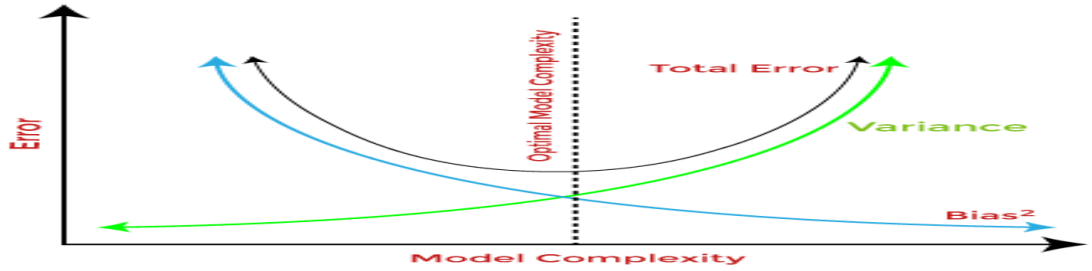
High variance can be identified if the model has low training error and high test error.

High Bias can be identified if the model has high training error and the test error is almost similar to training error.



Bias-Variance Trade-Off

While building the machine learning model, it is really important to take care of bias and variance in order to avoid overfitting and underfitting in the model. If the model is very simple with fewer parameters, it may have low variance and high bias. Whereas, if the model has a large number of parameters, it will have high variance and low bias. So, it is required to make a balance between bias and variance errors, and this balance between the bias error and variance error is known as **the Bias-Variance trade-off.**



For an accurate prediction of the model, algorithms need a low variance and low bias. But this is not possible because bias and variance are related to each other:

- If we decrease the variance, it will increase the bias.
- If we decrease the bias, it will increase the variance.

Bias-Variance trade-off is a central issue in Machine learning. Ideally, we need a model that accurately captures the regularities in training data and simultaneously generalizes well with the unseen dataset. Unfortunately, doing this is not possible simultaneously. Because a high variance algorithm may perform well with training data, but it may lead to overfitting to noisy data. Whereas, high bias algorithm generates a much simple model that may not even capture important regularities in the data. So, we need to find a **sweet spot** between bias and variance to make an optimal model. Hence, the Bias-Variance trade-off is about finding the sweet spot to make a balance between bias and variance errors.

V. Precision and Recall

While building any machine learning model, the first thing that comes to our mind is how we can build an accurate & 'good fit' model and what the challenges is that will come during the entire procedure. Precision and Recall are the two most important but confusing concepts in Machine Learning. They are performance metrics used for pattern recognition and classification in machine learning. These concepts are essential to build a perfect machine learning model which gives more precise and accurate results. Some of the models in machine learning require more precision and some model requires more recall. So, it is important to know the balance between Precision and recall or, simply, **precision-recall trade-off**.

Precision and recall are the two most confusing but important concepts in machine learning that lots of professionals face during their entire data science & machine learning career. But before starting, first, we need to understand the **confusion matrix** concept. So, let's start with the quick introduction of Confusion Matrix in Machine Learning.

Confusion Matrix in Machine Learning

Confusion Matrix helps us to display the performance of a model or how a model has made its prediction in Machine Learning. Confusion Matrix helps us to visualize the point where our model gets confused in discriminating two classes. It can be understood well through a 2×2 matrix where the row represents the **actual truth labels**, and the column represents **the predicted labels.**

This matrix consists of 4 main elements that show different metrics to count a number of correct and incorrect predictions. Each element has two words either as follows:

- True or False
- Positive or Negative

If the predicted and truth labels match, then the prediction is said to be correct, but when the predicted and truth labels are mismatched, then the prediction is said to be incorrect. Further, positive and negative represents the predicted labels in the matrix. There are four metrics combinations in the confusion matrix.

- **True Positive:** This combination tells us how many times a model correctly classifies a positive sample as Positive?
- **False Negative:** This combination tells us how many times a model incorrectly classifies a positive sample as Negative?
- **False Positive:** This combination tells us how many times a model incorrectly classifies a negative sample as Positive?
- **True Negative:** This combination tells us how many times a model correctly classifies a negative sample as Negative?

What is Precision?

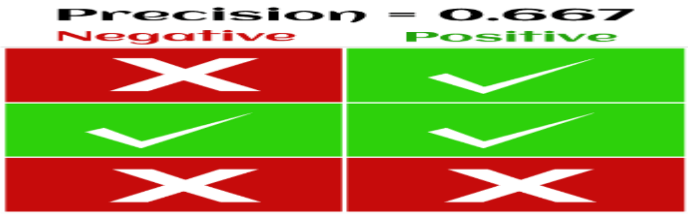
Precision *is defined as the* ratio of correctly classified positive samples (True Positive) to a total number of classified positive samples (either correctly or incorrectly).

1. Precision = True Positive/True Positive + False Positive
   Precision = TP/TP+FP
   - TP- True Positive, FP- False Positive

Hence, precision helps us to visualize the reliability of the machine learning model in classifying the model as positive**.**

Examples to calculate the Precision in the machine learning model

Case 1- In the below-mentioned scenario, the model correctly classified two positive samples while incorrectly classified one negative sample as positive. Hence, according to precision formula;



Precision = TP/TP+FP= 2/2+1 = 2/3 = 0.667

Case 2- In this scenario, we have three Positive samples that are correctly classified, and one Negative sample is incorrectly classified.

Put TP =3 and FP =1 in the precision formula, we get;

Precision = TP/TP+FP = 3/3+1 = 3/4 = 0.75

Case 3- In this scenario, we have three Positive samples that are correctly classified but no Negative sample which is incorrectly classified.

Put TP =3 and FP =0 in precision formula, we get=**Precision = TP/TP+FP** = 3/3+0 = 3/3 = 1

Hence, in the last scenario, we have a precision value of 1 or 100% when all positive samples are classified as positive, and there is no any Negative sample that is incorrectly classified.

What is Recall?

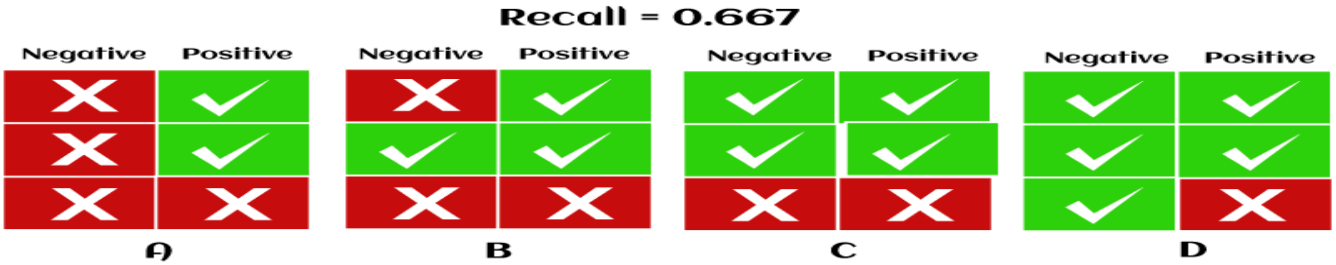
The recall is calculated as the ratio between the numbers of Positive samples correctly classified as Positive to the total number of Positive samples. The *recall measures the model's ability to detect positive samples*. The higher the recall, the more positive samples detected.

1. Recall = True Positive/True Positive + False Negative
2. Recall = TP/TP+FN
    - TP- True Positive, FN- False Negative

Unlike Precision, Recall is independent of the number of negative sample classifications. Further, if the model classifies all positive samples as positive, then Recall will be 1.

Examples to calculate the Recall in the machine learning model

Example 1- Let's understand the calculation of Recall with four different cases where each case has the same Recall as 0.667 but differs in the classification of negative samples. See how:



In this scenario, the classification of the negative sample is different in each case. Case A has two negative samples classified as negative, and case B have two negative samples classified as

negative; case C has only one negative sample classified as negative, while case D does not classify any negative sample as negative. However, recall is independent of how the negative samples are classified in the model; hence, we can neglect negative samples and only calculate all samples that are classified as positive.

In the above image, we have only two positive samples that are correctly classified as positive while only 1 negative sample that is correctly classified as negative. Hence, true positivity rate is 2 and while false negativity rate is 1. Then recall will be:

1. Recall = True Positive/True Positive + False Negative = TP/TP+FN =2/(2+1) =2/3 =0.667

Example-2: Now, we have another scenario where all positive samples are classified correctly as positive. Hence, the True Positive rate is 3 while the False Negative rate is 0.

Recall = TP/TP+FN = 3/(3+0) =3/3 =1

If the recall is 100%, then it tells us the model has detected all positive samples as positive and neglects how all negative samples are classified in the model. However, the model could still have so many samples that are classified as negative but recall just neglect those samples, which results in a high False Positive rate in the model.

Example-3: In this scenario, the model does not identify any positive sample that is classified as positive. All positive samples are incorrectly classified as Negative. Hence, the true positive rate is 0, and the False Negative rate is 3. Then Recall will be:

Recall = TP/TP+FN = 0/(0+3) =0, This means the model has not correctly classified any Positive Samples.

Difference between Precision and Recall in Machine Learning

Precision	Recall
It helps us to measure the ability to classify positive samples in the model.	It helps us to measure how many positive samples were correctly classified by the ML model.
While calculating the Precision of a model, we should	While calculating the Recall of a model, we only need

consider both Positive as well as Negative samples that are classified.	all positive samples while all negative samples will be neglected.
When a model classifies most of the positive samples correctly as well as many false-positive samples, then the model is said to be a high recall and low precision model.	When a model classifies a sample as Positive, but it can only classify a few positive samples, then the model is said to be high accuracy, high precision, and low recall model.
The precision of a machine learning model is dependent on both the negative and positive samples.	Recall of a machine learning model is dependent on positive samples and independent of negative samples.
In Precision, we should consider all positive samples that are classified as positive either correctly or incorrectly.	The recall cares about correctly classifying all positive samples. It does not consider if any negative sample is classified as positive.

Why use Precision and Recall in Machine Learning models?

This question is very common among all machine learning engineers and data researchers. The use of Precision and Recall varies according to the type of problem being solved.

- If there is a requirement of classifying all positive as well as Negative samples as Positive, whether they are classified correctly or incorrectly, then use Precision.
- Further, on the other end, if our goal is to detect only all positive samples, then use Recall. Here, we should not care how negative samples are correctly or incorrectly classified the samples.

Conclusion:

In this lecture, we have discussed various performance metrics such as confusion matrix, Precision, and Recall for binary classification problems of a machine learning model. Also, we have seen various examples to calculate Precision and Recall of a machine learning model and when we should use precision, and when to use Recall.