1. Metrics to evaluate Model Performance

Regression Model

1) R^2
2) **RMSE (Root Mean Square Error)**

RMSE is the most popular evaluation metric used in regression problems. It follows an assumption that error is unbiased and follow a normal distribution. RMSE is highly affected by outlier values. Hence, make sure you've removed outliers from your data set prior to using this metric.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

Classification Model

1) **Confusion Matrix**

A confusion matrix is an N X N matrix, where N is the number of classes being predicted. For the problem in hand, we have N=2, and hence we get a 2 X 2 matrix. Here are a few definitions, you need to remember for a confusion matrix:

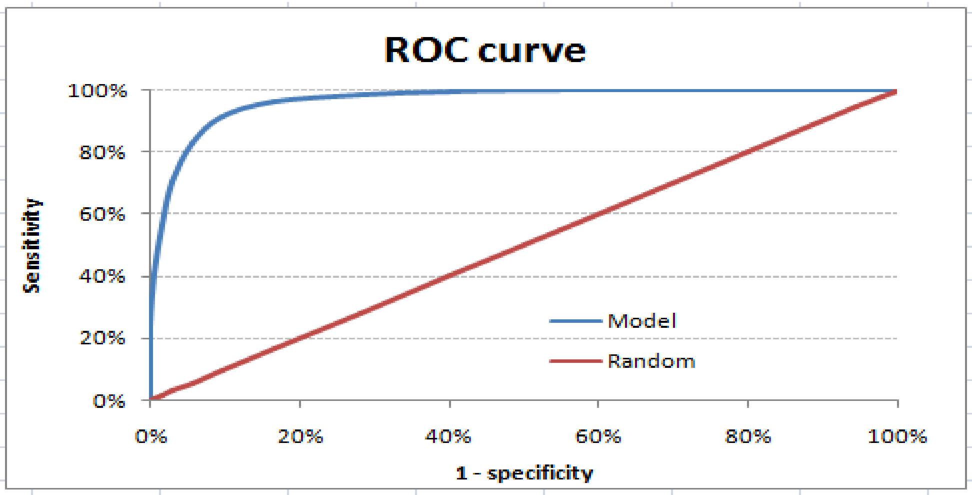
- **Accuracy**: the proportion of the total number of predictions that were correct.
- **Positive Predictive Value or Precision**: the proportion of positive cases that were correctly identified.
- **Negative Predictive Value**: the proportion of negative cases that were correctly identified.
- **Sensitivity or Recall**: the proportion of actual positive cases which are correctly identified.
- **Specificity**: the proportion of actual negative cases which are correctly identified.

For instance, in a pharmaceutical company, they will be more concerned with minimal wrong positive diagnosis. Hence, they will be more concerned about high Specificity. On the other hand, an attrition model will be more concerned with Sensitivity. Confusion matrix are generally used only with class output models.

Accuracy works well only if there are equal number of samples belonging to each class. For example, consider that there are 98% samples of class A and 2% samples of class B in our training set. Then our model can easily get **98% training accuracy** by simply predicting every training sample belonging to class A. When the same model is tested on a test set with 60% samples of class A and 40% samples of class B, then the **test accuracy would drop down to 60%.** Classification Accuracy is great, but gives us the false sense of achieving high accuracy.

2) Area under the ROC Curve (AUC -ROC)

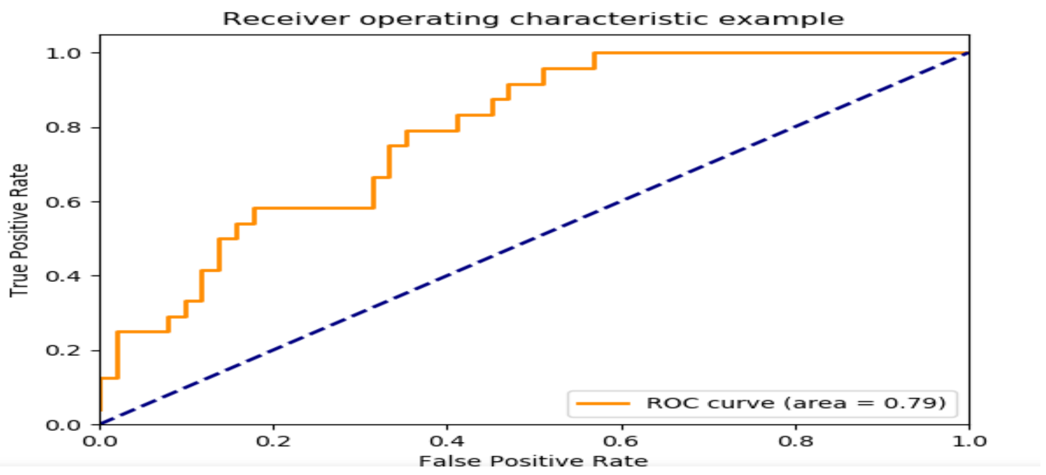
The ROC curve is the plot between sensitivity and (1- specificity). (1- specificity) is also known as false positive rate and sensitivity is also known as True Positive rate. Following is the ROC curve for the case in hand.



The area of entire square is 1*1 = 1. Hence AUC itself is the ratio under the curve and the total area. For the case in hand, we get AUC ROC as 96.4%. Following are a few thumb rules:

- .90-1 = excellent (A)
- .80-.90 = good (B)
- .70-.80 = fair (C)
- .60-.70 = poor (D)
- .50-.60 = fail (F)

False Positive Rate and True Positive Rate both have values in the range [0, 1]. FPR and TPR both are computed at threshold values such as (0.00, 0.02, 0.04, …., 1.00) and a graph is drawn. AUC is the area under the curve of plot False Positive Rate vs True Positive Rate at different points in [0, 1].



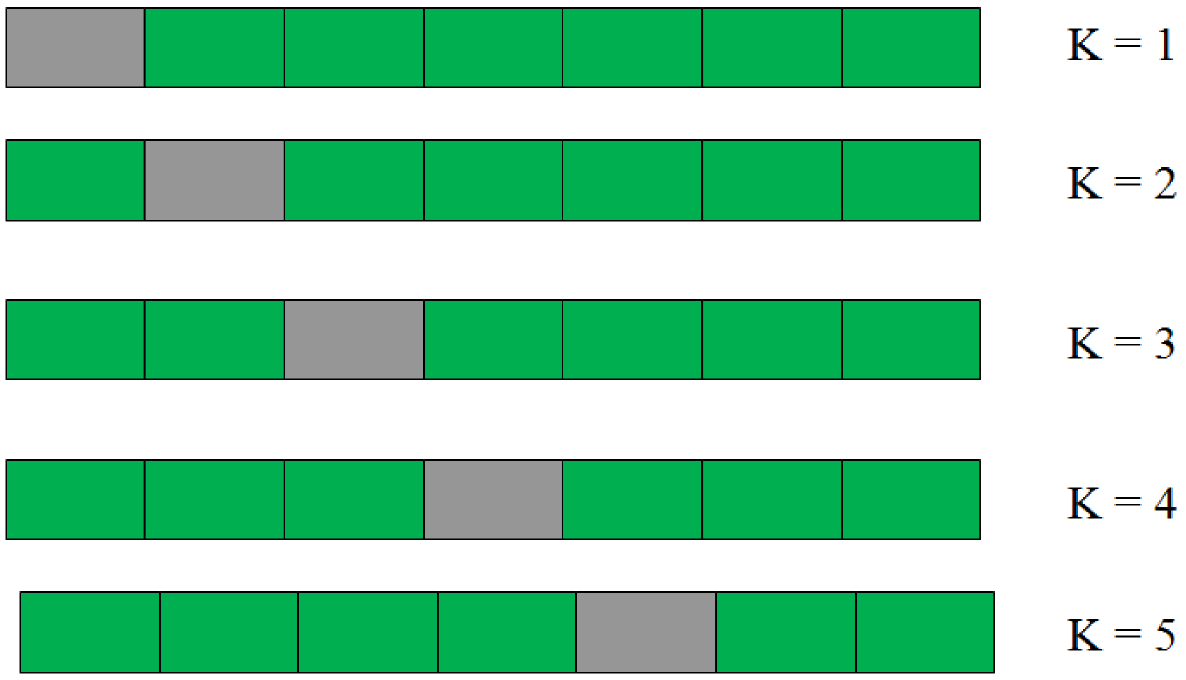
As evident, AUC has a range of [0, 1]. The greater the value, the better is the performance of our model. Higher the AUC, better the model.

3) Cross Validation

Cross validation isn't a really an evaluation metric which is used openly to communicate model accuracy. But, the result of cross validation provides good enough intuitive result to generalize the performance of a model.

It simply says, try to leave a sample on which you do not train the model and test the model on this sample before finalizing the model. If we make a 50:50 split of training population and the train on first 50 and validate on rest 50. Then, we train on the other 50, test on first 50. This way we train the model on the entire population, however on 50% in one go. This reduces bias because of sample selection to some extent but gives a smaller sample to train the model on. This approach is known as 2-fold cross validation.

K-fold Cross validation



This is a 5-fold cross validation.

Here's what goes on behind the scene: we divide the entire population into 5 equal samples. Now we train models on 4 samples (Green boxes) and validate on 1 sample (grey box). Then, at the second iteration we train the model with a different sample held as validation. In 5 iterations, we have basically built model on each sample and held each of them as validation. This is a way to reduce the selection bias and reduce the variance in prediction power. Once we have all the 5 models, we take average of the error terms to find which of the models is best.

K-fold cross validation is widely used to check whether a model is an overfit or not. If the performance metrics at each of the k times modelling are close to each other and the mean of metric is highest.

4) **F1 Score**

F1 Score is used to measure a test's accuracy. F1 Score is the Harmonic Mean between precision and recall. The range for F1 Score is [0, 1]. It tells you how precise your classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances).

High precision but lower recall, gives you an extremely accurate, but it then misses a large number of instances that are difficult to classify. The greater the F1 Score, the better is the performance of our model. Mathematically, it can be expressed as:

$$F1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}}$$

F1 Score tries to find the balance between precision and recall.

5) **Mean Absolute Error**

Mean Absolute Error is the average of the difference between the Original Values and the Predicted Values. It gives us the measure of how far the predictions were from the actual output. However, they don't give us any idea of the direction of the error i.e. whether we are under predicting the data or over predicting the data. Mathematically, it is represented as:

$$MeanAbsoluteError = \frac{1}{N} \sum_{j=1}^{N} |y_j - \hat{y}_j|$$

6) **Mean Squared Error**

Mean Squared Error (MSE) is quite similar to Mean Absolute Error, the only difference being that MSE takes the average of the **square** of the difference between the original values and the predicted values. The advantage of MSE being that it is easier to compute the gradient, whereas Mean Absolute Error requires complicated linear programming tools to compute the gradient. As, we take square of the error, the effect of larger errors become more pronounced then smaller error, hence the model can now focus more on the larger errors.

$$MeanSquaredError = \frac{1}{N} \sum_{j=1}^{N} (y_j - \hat{y}_j)^2$$

When to use Accuracy:

Accuracy is a good measure when the target variable classes in the data are nearly balanced.
Ex: 60% classes in our fruit's images data are apple and 40% are oranges.
A model which predicts whether a new image is Apple or an Orange, 97% of times correctly is a very good measure in this example.

When NOT to use Accuracy:

Accuracy should NEVER be used as a measure when the target variable classes in the data are a majority of one class.
Ex: In our cancer detection example with 100 people, only 5 people has cancer. Let's say our model is very bad and predicts every case as No Cancer. In doing so, it has classified those 95 non-cancer patients correctly and 5 cancerous patients as Non-cancerous. Now even though the model is terrible at predicting cancer, the accuracy of such a bad model is also 95%.

When to use Precision and When to use Recall

It is clear that recall gives us information about a classifier's performance with respect to false negatives (how many did we miss), while precision gives us information about its performance with respect to false positives (how many did we caught).

Precision is about being precise. So even if we managed to capture only one cancer case, and we captured it correctly, then we are 100% precise.

Recall is not so much about capturing cases correctly but more about capturing all cases that have "cancer" with the answer as "cancer". So, if we simply always say every case as "cancer", we have 100% recall.

So basically, if we want to focus more on minimising False Negatives, we would want our Recall to be as close to 100% as possible without precision being too bad and if we want to focus on minimising False positives, then our focus should be to make Precision as close to 100% as possible.

F1 Score

We don't really want to carry both Precision and Recall in our pockets every time we make a model for solving a classification problem. So, it's best if we can get a single score that kind of represents both Precision (P) and Recall (R).

One way to do that is simply taking their arithmetic mean. i.e. (P + R) / 2 where P is Precision and R is Recall. But that's pretty bad in some situations. Suppose we have 100 credit card transactions, of which 97 are legit and 3 are fraud and let's say we came up a model that predicts everything as fraud.

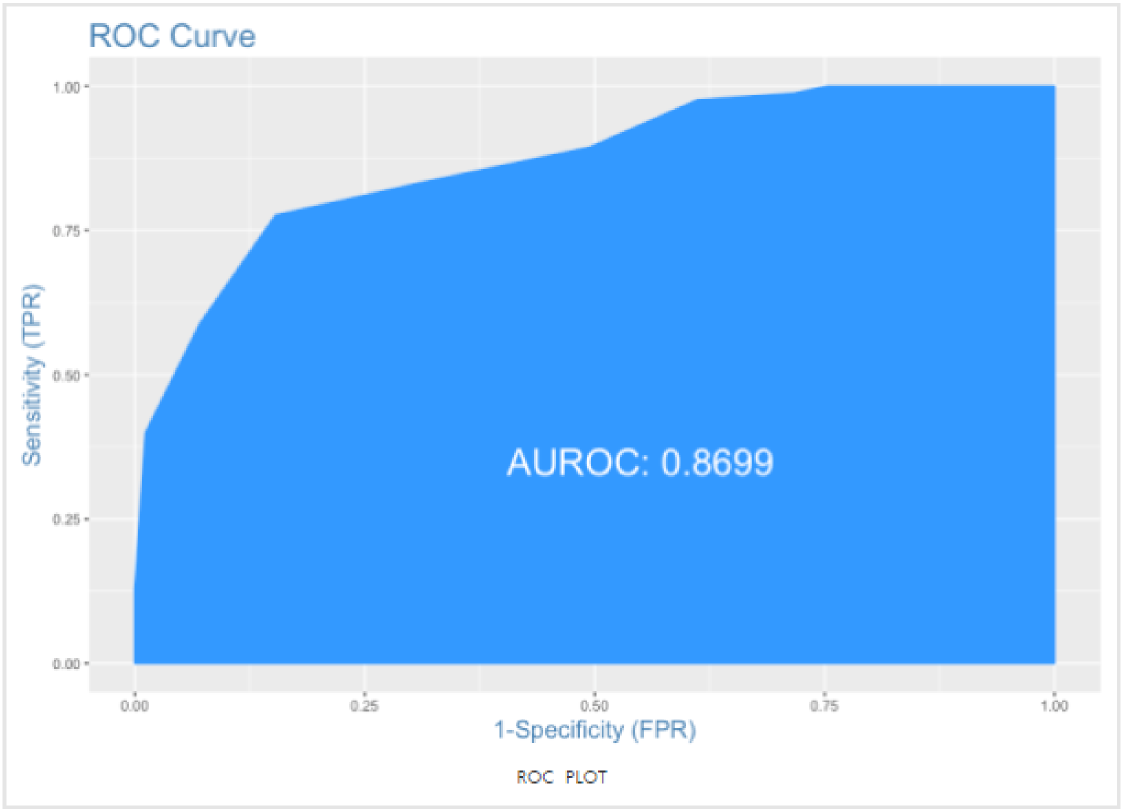
Now, if we simply take arithmetic mean of both, then it comes out to be nearly 51%. We shouldn't be giving such a moderate score to a terrible model since it's just predicting every transaction as fraud.

So, we need something more balanced than the arithmetic mean and that is harmonic mean. Harmonic mean is kind of an average when x and y are equal. But when x and y are different, then it's closer to the smaller number as compared to the larger number.

A high precision score gives more confidence to the model's capability to classify 1's. Combining this with Recall gives an idea of how many of the total 1's it was able to cover.

A good model should have a good precision as well as a high recall. So ideally, I want to have a measure that combines both these aspects in one single metric – **the F1 Score**.

F1 Score = (2 * Precision * Recall) / (Precision + Recall)



Interpreting the ROC plot is very different from a regular line plot. Because, though there is an X and a Y-axis, you don't read it as: for an X value of 0.25, the Y value is .9.

Instead, what we have here is a line that traces the probability cutoff from 1 at the bottom-left to 0 in the top right. This is a way of analyzing how the sensitivity and specificity perform for the full range of probability cutoffs, that is from 0 to 1.

Ideally, if you have a perfect model, all the events will have a probability score of 1 and all non-events will have a score of 0. For such a model, the area under the ROC will be a perfect 1.

So, if we trace the curve from bottom left, the value of probability cutoff decreases from 1 towards 0. If you have a good model, more of the real events should be predicted as events, resulting in high sensitivity and low FPR. In that case, the curve will rise steeply covering a large area before reaching the top-right.

Therefore, the larger the area under the ROC curve, the better is your model. The ROC curve is the only metric that measures how well the model does for different values of prediction probability cutoffs.

--

Noise: In predictive modeling, "Noise," refers to the irrelevant information or randomness in a dataset.

----

2. <u>How to Detect Overfitting</u>

Over-fitting is nothing but when you model become highly complex that it starts capturing noise also. This 'noise' adds no value to model, but only inaccuracy.

- We can split our initial dataset into separate training and test subsets. **If our model does much better on the training set than on the test set, then we're likely overfitting.** For example, it would be a big red flag if our model saw 99% accuracy on the training set but only 55% accuracy on the test set.

- High value of R^2

- Overfitting is suspect when the model accuracy is high with respect to the data used in training the model but drops significantly with new data. Effectively the model knows the training data well but does not generalize. This makes the model useless for purposes such as prediction.

- A simple way to detect this effect in practice is cross-validation. This attempts to examine the trained model with new data set to check its predictive accuracy. Given a dataset, some portion of this is held out (say 30%) while the rest is used in training the model. Upon training the model, the held-out data is then used to check the accuracy compared to the accuracy of derived from the data used in training. A significant variance in these two flags overfitting.

- You can detect overfitting by determining whether your model fits new data as well as it fits the data used to estimate the model. In statistics, we call this cross-validation, and it often involves partitioning your data.

- We built a model on the original dataset, and it predicts outcomes with 99% accuracy, but when we run the model on a new ("unseen") dataset of resumes, we only get 50% accuracy. Our model doesn't generalize well from our training data to unseen data.

- When dataset has too many input features given a smaller number of observations.

<u>Goodness of Fit</u>

In statistics, goodness of fit refers to how closely a model's predicted values match the observed (true) values. A model that has learned the noise instead of the signal is considered "overfit" because it fits the training dataset but has poor fit with new datasets. In predictive modeling, you can think of the "signal" as the true underlying pattern that you wish to learn from the data. Noise," on the other hand, refers to the irrelevant information or randomness in a dataset.

<u>Cross-validation</u>

One way to get around wasting a large part of the data to validation and test is to use cross-validation (CV) which estimates the generalized performance using the same data as is used to train the model. The idea behind cross-validation is to split the dataset up into a certain number of subsets, and then use each of these subsets as held out test sets in turn while using the rest of the data to train the model. Averaging the metric over all the folds will give you an estimate of the model performance. The final model is then generally trained using all data.

----

3. <u>How to Prevent Overfitting</u>

- Train with more data
- Remove features
- **Pruning:** Pruning is used extensively while building CART models. It simply removes the nodes which add little predictive power for the problem in hand
- **Early Stopping**: Early stopping rules provide guidance as to how many iterations can be run before the learner begins to over-fit. If the training algorithm is iterative (such as gradient descent), monitor the performance in a validation set and use early stopping when this performance starts to drop (or becomes stable).
- **Regularization**: Regularization "refers to a process of introducing additional information in order to solve an ill-posed problem or to prevent overfitting.
- **Data augmentation**: Data augmentation is the process of using the data you currently have and modifying it in a realistic but randomised way, to increase the variety of data seen during training. As an example, for images, slightly rotating, zooming, and/or translating the image will result in the same content, but with a different framing

➢ Bagging and Boosting

Bagging attempts to reduce the chance overfitting complex models.

- It trains a large number of "strong" learners in parallel.
- A strong learner is a model that's relatively unconstrained.
- Bagging then combines all the strong learners together in order to "smooth out" their predictions.

Boosting attempts to improve the predictive flexibility of simple models.

- It trains a large number of "weak" learners in sequence.
- A weak learner is a constrained model (i.e. you could limit the max depth of each decision tree).
- Each one in the sequence focuses on learning from the mistakes of the one before it.
- Boosting then combines all the weak learners into a single strong learner.

Bagging uses complex base models and tries to "smooth out" their predictions, while boosting uses simple base models and tries to "boost" their aggregate complexity.

An overfit model result in misleading regression coefficients, p-values, and R-squared statistics.

Cross Validation

Cross-validation is a technique used to protect against overfitting in a predictive model, particularly in a case where the amount of data may be limited. In cross-validation, you make a fixed number of folds (or partitions) of the data, run the analysis on each fold, and then average the overall error estimate. Cross-validation is a statistical method used to estimate the skill of machine learning models. Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample.

The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation.

The general procedure is as follows:

1. Shuffle the dataset randomly.
2. Split the dataset into k groups
3. For each unique group:
1. Take the group as a hold out or test data set
2. Take the remaining groups as a training data set
3. Fit a model on the training set and evaluate it on the test set

4. Retain the evaluation score and discard the model
4. Summarize the skill of the model using the sample of model evaluation scores
   Yes! That method is known as "**k-fold cross validation**".

CV is easy to follow and implement. Below are the steps for it:

1. Randomly split your entire dataset into k-folds"
2. For each k-fold in your dataset, build your model on k – 1 folds of the dataset. Then, test the model to check the effectiveness for *kth* fold
3. Record the error you see on each of the predictions
4. Repeat this until each of the k-folds has served as the test set
5. The average of your k recorded errors is called the cross-validation error and will serve as your performance metric for the model

Ex: Say you choose k=5 in for k-fold cross validation. Split your data into 5 equal parts. You train your algorithm on 4/5 = 80% of the data, then test on the remaining 1/5 = 20%. You do that five times, each selecting a different testing data set. Then you average the accuracy rate, and compute standard deviation of accuracy rate, etc.

The goal of cross-validation is to test the model's ability to predict new data that was not used in estimating it, in order to flag problems like overfitting or selection bias and to give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem).

Regularization

Regularization is a process of introducing additional information in order to solve an ill-posed problem or to prevent overfitting.

Early stopping

Early stopping is a form of regularization used to avoid overfitting when training a learner with an iterative method, such as gradient descent, boosting. Such methods update the learner so as to make it better fit the training data with each iteration. Up to a point, this improves the learner's performance on data outside of the training set. Past that point, however, improving the learner's fit to the training data comes at the expense of increased generalization error. Early stopping rules provide guidance as to how many iterations can be run before the learner begins to over-fit. Early stopping rules have been employed in many different machine learning methods, with varying amounts of theoretical foundation.

Definitions of Train, Validation and Test Datasets

Training Dataset: The sample of data used to fit the model. The model is initially fit on a training dataset. A training dataset is a dataset of examples used for learning, that is to fit the parameters (e.g., weights) of, for example, a classifier.

Validation Dataset: The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The validation set is used to evaluate a given model, but this is for frequent evaluation. We use this data to fine-tune the model hyperparameters. This data set is used to minimize overfitting.

A validation dataset is a dataset of examples used to tune the hyperparameters (i.e. the architecture) of a classifier. In order to avoid overfitting, when any classification parameter needs to be adjusted, it is necessary to have a validation dataset in addition to the training and test datasets.

Successively, the fitted model is used to predict the responses for the observations in a second dataset called the validation dataset. The validation dataset provides an unbiased evaluation of a model fit on the training dataset while tuning the model's hyperparameters.

If the accuracy over the training data set increases, but the accuracy over then validation data set stays the same or decreases, then you're overfitting your model and you should stop training.

Test Dataset: The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset. The Test dataset provides the gold standard used to evaluate the model. It is only used once a model is completely trained (using the train and validation sets). Finally, the test dataset is a dataset used to provide an unbiased evaluation of a final model fit on the training dataset. When the data in the test dataset has never been used in training (for example in cross-validation), the test dataset is also called a **holdout dataset**.

A test dataset is a dataset that is independent of the training dataset, but that follows the same probability distribution as the training dataset. If a model fit to the training dataset also fits the test dataset well, minimal overfitting has taken place (see figure below). A better fitting of the training dataset as opposed to the test dataset usually points to overfitting.

Test data set is used only for testing the final solution in order to confirm the actual predictive power of the model.

Training dataset is used to train the candidate algorithms, the validation dataset is used to compare their performances and decide which one to take and, finally, the test dataset is used to obtain the performance characteristics such as accuracy, sensitivity, specificity, F-measure, and so on.

1. The training set represents about **80%** of the available data, and is used to train the model.
2. The test set consists of the remaining **20%** of the dataset, and is used to test the **accuracy** of the model on data it has **never seen before**.

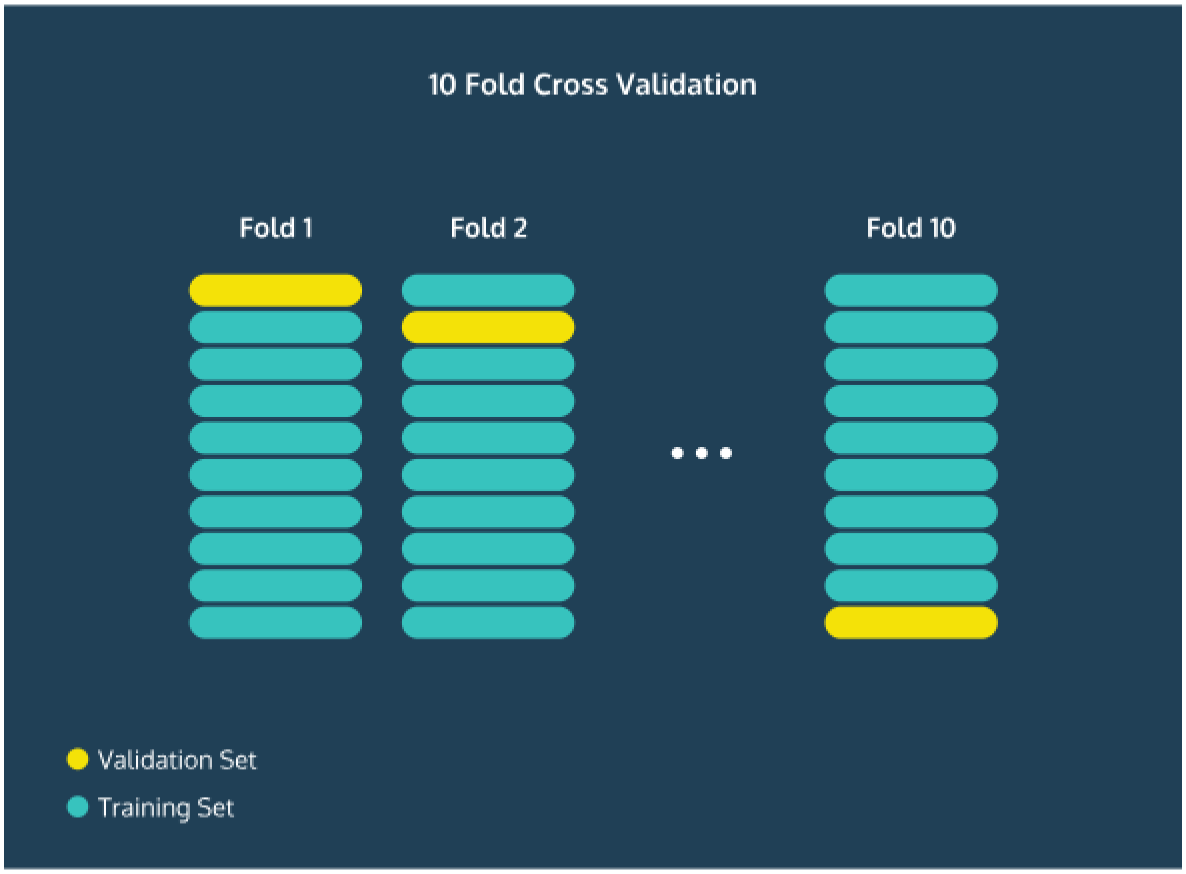
Training Set Vs Validation Set

The training set is the data that the algorithm will learn from. After training using the training set, the points in the validation set are used to compute the accuracy or error of the classifier. The key insight here is that we know the true labels of every point in the validation set, but we're temporarily going to pretend like we don't. We can use every point in the validation set as input to our classifier. We'll then receive a classification for that point. We can now peek at the true label of the validation point and see whether we got it right or not. If we do this for every point in the validation set, we can compute the validation error!

Validation error might not be the only metric we're interested in. A better way of judging the effectiveness of a machine learning algorithm is to compute its precision, recall, and F1 score.

N-Fold Cross-Validation

Sometimes your dataset is so small, that splitting it 80/20 will still result in a large amount of variance. One solution to this is to perform **N-Fold Cross-Validation**. The central idea here is that we're going to do this entire process N times and average the accuracy. For example, in 10-fold cross-validation, we'll make the validation set the first 10% of the data and calculate accuracy, precision, recall and F1 score. We'll then make the validation set the second 10% of the data and calculate these statistics again. We can do this process 10 times, and every time the validation set will be a different chunk of the data. If we then average all of the accuracies, we will have a better sense of how our model does on average.



Test Dataset

Once you're happy with your model's performance, it is time to introduce the test set. This is part of your data that you partitioned away at the very start of your experiment. It's meant to be a substitute for the data in the real world that you're actually interested in classifying. It functions very similarly to the validation set, except you never touched this data while building or tuning your model. By finding the accuracy, precision, recall, and F1 score on the test set, you get a good understanding of how well your algorithm will do in the real world.

Cost Function

"What does the machine (i.e. the statistical model) actually learn?"

This will vary from model to model, but in simple terms the model learns a function *f* such that *f*(*X*) maps to *y*. Put differently, the model learns how to take *X* (i.e. features, or, more traditionally, independent variable(s)) in order to predict *y* (the target, response or more traditionally the dependent variable).

In the case of the simple linear regression (***y* ~ b0 + b1 * *X*** where *X* is one column/variable) the model "learns" (read: estimates) two parameters;

- b0: the bias (or more traditionally the intercept); and,
- b1: the slope

The bias is the level of *y* when *X* is 0 (i.e. the value of sales when advertising spend is 0) and the slope is the rate of predicted increase or decrease in *y* for each unit increase in *X* (i.e. how much do sales increase per pound spent on advertising). Both parameters are scalars (single values).

Once the model learns these parameters they can be used to compute estimated values of *y* given new values of *X*. In other words, you can use these learned parameters to predict values of *y* when you don't know what *y* is — hey presto, a predictive model.

There are several ways to learn the parameters of a LR model; one of the approaches is minimising a **cost function**.

In ML, cost functions are used to estimate how badly models are performing. Put simply, **a cost function is a measure of how wrong the model is in terms of its ability to estimate the relationship between X and y.** This is typically expressed as a difference or distance between the predicted value and the actual value. The cost function (you may also see this referred to as loss or error.) can be estimated by iteratively running the model to compare estimated predictions against "ground truth" — the known values of y.

The objective of a ML model, therefore, is to find parameters, weights or a structure that **minimises** the cost function. Your cost function might be the sum of squared errors over your training set. Gradient descent is a method for finding the minimum of a function of multiple variables. So, you can use gradient descent to minimize your cost function. we are using cost function to measure how close the predicted values are to their corresponding real values.

Ex: We also should consider that the weights of the trained model are responsible for accurately predicting the new values. Imagine that our model is y = 0.9*X + 0.1, the predicted value is nothing but (0.9*X+0.1) for different **Xs**. [0.9 and 0.1 in the equation are just random values to understand.]

So, by considering Y as real value corresponding to this x, the cost formula is coming to measure how close (0.9*X+0.1) is to Y.

We are responsible for finding the better weight (0.9 and 0.1) for our model to come up with a lowest cost (or closer predicted values to real ones).

Gradient descent is an optimization algorithm (we have some other optimization algorithms) and its responsibility is to find the minimum cost value in the process of trying the model with different weights or indeed, updating the weights. We first run our model with some initial weights and gradient descent updates our weights and find the cost of our model with those weights in thousands of iterations to find the minimum cost. One point is that gradient descent is not minimizing the weights, it is just updating them. This algorithm is looking for minimum cost.

Cost function is something is like at what cost you are building your model for a good model that cost should be minimum. To find the minimum cost function we use gradient descent method. That give value of coefficients to determine minimum cost function. For ex: **In other words, I want to minimize the difference between the prediction and the actual price of the houses.** Find me the values of this parameters that minimize the expression on blue, so that the sum of squared errors between the predicted value and the actual value is as small as possible (minimized). So, my cost function is also called squared error function.

<u>Difference between a cost function and a loss function</u>

The loss function (or error) is for a single training example, while the cost function is over the entire training set (or mini-batch for mini-batch gradient descent). Loss function is for one training example Cost function is the for the entire training set. Loss function computes the error for a single training example while cost function is the average of loss functions of the entire training set.

<u>Bias Variance:</u>

Bias is the difference between the expected or average prediction of our model and the correct value which we are trying to predict. Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.

Variance is the variability of model prediction for a given data point or a value which tells us spread of our data. Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn't seen before. As a result, such models perform very well on training data but has high error rates on test data.

High variance means that your estimator (or learning algorithm) **varies** a lot depending on the data that you give it. If your algorithm is able to fit your data **extremely well every single time** and even a single data point perturbation changes the algorithm a lot then the algorithm is having high variance. This type of high variance is called overfitting. Thus, usually overfitting is related to high variance.

To calculate the bias of a method used for many estimates, find the errors by subtracting each estimate from the actual or observed value. Add up all the errors and divide by the number of estimates to get the bias. If the errors add up to zero, the estimates were unbiased, and the method delivers unbiased results. If the estimates are biased, it may be possible to find the source of the bias, and eliminate it to improve the method.