1.R-squared or Residual Sum of Squares (RSS) which one of these two is a better measure of goodness of fit model in regression and why?

Answer:

R-squared (R^2) and Residual Sum of Squares (RSS) are both commonly used measures to assess the goodness of fit of a regression model, but they capture different aspects of model performance, and the choice between them depends on the context and what you want to evaluate.

1. **R-squared (R^2)**:

- R-squared is a measure of the proportion of the variance in the dependent variable (the response variable) that is explained by the independent variables (predictors) in your regression model.

- R-squared ranges from 0 to 1, with higher values indicating a better fit. A value of 1 means that the model perfectly explains the variance in the dependent variable, while a value of 0 means that the model provides no explanatory power.

- R-squared is a relative measure, and it does not provide information about the absolute goodness of fit or the quality of the model's predictions.

R-squared can be useful for comparing different models or assessing how much of the variation in the dependent variable can be attributed to the predictors. However, it has limitations. For example, it can be artificially inflated by adding more predictors to a model, even if those predictors do not have a meaningful relationship with the dependent variable.

2. **Residual Sum of Squares (RSS)**:

- RSS measures the total squared difference between the observed values of the dependent variable and the predicted values from the regression model. It quantifies the overall error or "residuals" in the model's predictions.

- A smaller RSS indicates a better fit because it means that the model's predictions are closer to the actual observed values.

RSS is an absolute measure of the goodness of fit. It tells you how well the model fits the data in terms of minimizing prediction errors. Unlike R-squared, RSS does not provide information about the proportion of variance explained but gives you a direct measure of the model's prediction accuracy.

Which Measure to Use:

- R-squared is often used when you want to understand the proportion of variance explained by the model, especially in the context of comparing different models. It can provide insights into how well the predictors collectively contribute to explaining the variation in the dependent variable.

- RSS is useful when you want to evaluate the absolute goodness of fit, focusing on the magnitude of the prediction errors. If minimizing prediction errors is a primary concern, RSS is a more appropriate choice.

In practice, both measures can be valuable. R-squared provides a high-level summary of the explanatory power of your model, while RSS gives you a detailed view of the model's prediction accuracy. The choice between them should align with your specific goals and the questions you want to answer about your regression model.

2. What are TSS (Total Sum of Squares), ESS (Explained Sum of Squares) and RSS (Residual Sum of Squares) in regression. Also mention the equation relating these three metrics with each other.

Answer:

TSS (total sum of squares) is equal to **ESS** (explained sum of squares) plus **RSS** (residual sum of squares), we need to start with the definitions of these terms and then use some algebraic manipulations to arrive at the desired result.

Let us begin by defining the three terms:

TSS = $\sum(Yi - \bar{Y})^2$, where Yi is the actual value of the response variable for observation i, and $\bar{Y}$ is the mean of the response variable.

ESS = $\sum(\hat{Y}i - \bar{Y})^2$, where $\hat{Y}i$ is the predicted value of the response variable for observation i.

RSS = $\sum(Yi - \hat{Y}i)^2$, which is the sum of squared differences between the actual and predicted values of the response variable.

Now, we can expand the terms in the definition of TSS using the definition of $\hat{Y}i$:

TSS = $\sum(Yi - \bar{Y})^2 = \sum[(Yi - \hat{Y}i) + (\hat{Y}i - \bar{Y})]^2 = \sum(Yi - \hat{Y}i)^2 + \sum(\hat{Y}i - \bar{Y})^2 + 2\sum(Yi - \hat{Y}i)(\hat{Y}i - \bar{Y})$

Next, we can use the definition of RSS to simplify the first term on the right-hand side:

$\sum(Yi - \hat{Y}i)^2 = RSS$

Similarly, we can use the definition of ESS to simplify the second term:

$\sum(\hat{Y}i - \bar{Y})^2 = ESS$

Now, we need to simplify the third term using some algebraic manipulations. We can start by expanding the product:

$2\sum(Yi - \hat{Y}i)(\hat{Y}i - \bar{Y}) = 2\sum(Yi\hat{Y}i - Yi\bar{Y} - \hat{Y}i\bar{Y} + \hat{Y}i^2)$

Then, we can use the fact that the sum of the residuals (Yi - Ŷi) is zero, which can be shown as follows:

$$\sum(Yi - \hat{Y}i) = \sum Yi - \sum \hat{Y}i = n\bar{Y} - n\bar{Y} = 0$$

Using this fact, we can simplify the third term:

$$2\sum(Yi - \hat{Y}i)(\hat{Y}i - \bar{Y}) = 2\sum(Yi\hat{Y}i - \hat{Y}i\bar{Y}) = 2(\sum Yi\hat{Y}i - \sum \hat{Y}i\bar{Y}) = 2(\sum \hat{Y}iYi - n\bar{Y}^2) = 2(ESS - n(\bar{Y} - \hat{Y})^2)$$

where $\hat{Y}$ is the sample mean of the predicted values, which is equal to $\bar{Y}$.

Substituting these results back into the equation for TSS, we get:

$$TSS = RSS + ESS + 2(ESS - n(\bar{Y} - \hat{Y})^2) = RSS + 2ESS - 2n(\bar{Y} - \hat{Y})^2 = RSS + 2ESS - 2n(\bar{Y} - \bar{Y})^2 = RSS + 2ESS - 0 = RSS + ESS$$

Therefore, we have shown that TSS is equal to ESS plus RSS.

3. What is the need of regularization in machine learning?

Answer:

Regularization is one of the most important concepts of machine learning. It is a technique to prevent the model from overfitting by adding extra information to it.

Sometimes the machine learning model performs well with the training data but does not perform well with the test data. It means the model is not able to predict the output when deals with unseen data by introducing noise in the output, and hence the model is called overfitted. This problem can be deal with the help of a regularization technique.

This technique can be used in such a way that it will allow to maintain all variables or features in the model by reducing the magnitude of the variables. Hence, it maintains accuracy as well as a generalization of the model.

It mainly regularizes or reduces the coefficient of features toward zero. In simple words, "*In regularization technique, we reduce the magnitude of the features by keeping the same number of features.*"

There are mainly two techniques of regularization:

1.Ridge Regression

Ridge regression is one of the types of linear regression in which a small amount of bias is introduced so that we can get better long-term predictions

2.Lasso Regression

Lasso regression is another regularization technique to reduce the complexity of the model. It stands for Least Absolute and Selection Operator.

It is similar to the Ridge Regression except that the penalty term contains only the absolute weights instead of a square of weights.

4. What is Gini–impurity index?

Answer:

Gini Impurity is a measurement used to build Decision Trees to determine how the features of a dataset should split nodes to form the tree. More precisely, the Gini Impurity of a dataset is a number between 0-0.5, which indicates the likelihood of new, random data being misclassified if it were given a random class label according to the class distribution in the dataset.

For example, say you want to build a classifier that determines if someone will default on their credit card. You have some labeled data with features, such as bins for age, income, credit rating, and whether or not each person is a student. To find the best feature for the first split of the tree – the root node – you could calculate how poorly each feature divided the data into the correct class, default ("yes") or didn't default ("no"). This calculation would measure the **impurity** of the split, and the feature with the lowest impurity would determine the best feature for splitting the current node. This process would continue for each subsequent node using the remaining features

The Gini Index or Gini Impurity is calculated by subtracting the sum of the squared probabilities of each class from one. It favours mostly the larger partitions and are very simple to implement. In simple terms, it calculates the probability of a certain randomly selected feature that was classified incorrectly.

The Gini Index varies between 0 and 1, where 0 represents purity of the classification and 1 denotes random distribution of elements among various classes. A Gini Index of 0.5 shows that there is equal distribution of elements across some classes.

Mathematically, The Gini Index is represented by

$$G = \sum_{i=1}^{C} p(i) * (1 - p(i))$$

The Gini Index works on categorical variables and gives the results in terms of “success” or “failure” and hence performs only binary split. It isn’t computationally intensive as its counterpart – Information Gain. From the Gini Index, the value of another parameter named Gini Gain is calculated whose value is maximised with each iteration by the Decision Tree to get the perfect CART

5. Are unregularized decision-trees prone to overfitting? If yes, why?

Answer:

Decision trees are prone to overfitting, especially when a tree is particularly deep. This is due to the amount of specificity we look at leading to smaller sample of events that meet the previous assumptions. This small sample could lead to unsound conclusions. An example of this could be predicting if the Boston Celtics will beat the Miami Heat in tonight’s basketball game. The first level of the tree could ask if the Celtics are playing home or away. The second level might ask if the Celtics have a higher win percentage than their opponent, in this case the Heat. The third level asks if the Celtic’s leading scorer is playing? The fourth level asks if the Celtic’s second leading scorer is playing. The fifth level asks if the Celtics are traveling back to the east coast from 3 or more consecutive road games on the west coast. While all of these questions may be relevant, there may only be two previous games where the conditions of tonights game were met. Using only two games as the basis for our classification would not be adequate for an informed decision. One way to combat this issue is by setting a max depth. This will limit our risk of overfitting; but as always, this will be at the expense of error due to bias. Thus if we set a max depth of three, we would only ask if the game is home or away, do the Celtics have a higher winning percentage than their opponent, and is their leading scorer playing. This is a simpler model with less variance sample to sample but ultimately will not be a strong predictive model.

Ideally, we would like to minimize both error due to bias and error due to variance. Enter random forests. Random forests mitigate this problem well. A random forest is simply a collection of decision trees whose results are aggregated into one final result. Their ability to limit overfitting without substantially increasing error due to bias is why they are such powerful models.

6. What is an ensemble technique in machine learning?

Answer:

Ensemble Learning Techniques in Machine Learning, Machine learning models suffer bias and/or variance. Bias is the difference between the predicted value and actual value by the model. Bias is introduced when the model doesn't consider the variation of data and creates a simple model. The simple model doesn't follow the patterns of data, and hence the model gives errors in predicting training as well as testing data i.e. the model with high bias and high variance. You Can also read Covariance and Correlation In Machine Learning

When the model follows even random quirks of data, as pattern of data, then the model might do very well on training dataset i.e. it gives low bias, but it fails on test data and gives high variance.

Therefore, to improve the accuracy (estimate) of the model, ensemble learning methods are developed. Ensemble is a machine learning concept, in which several models are trained using machine learning algorithms. It combines low performing classifiers (also called as weak learners or base learner) and combine individual model prediction for the final prediction.

On the basis of type of base learners, ensemble methods can be categorized as homogeneous and heterogeneous ensemble methods. If base learners are same, then it is a homogeneous ensemble method. If base learners are different then it is a heterogeneous ensemble method.

Ensemble techniques are classified into three types:

1. Bagging: Bagging, also known as bootstrap aggregation, is an ensemble learning technique that combines the benefits of bootstrapping and aggregation to yield a stable model and improve the prediction performance of a machine learning model
2. Boosting: In boosting, we train a sequence of models. Each model is trained on a weighted training set. We assign weights based on the errors of the previous models in the sequence.
3. Stacking: In stacking, the predictions of base models are fed as input to a meta-model (or meta-learner). The job of the meta-model is to take the predictions of the base models and make a final prediction

7. What is the difference between Bagging and Boosting techniques?

Answer:

S.NO	Bagging	Boosting
1.	The simplest way of combining predictions that belong to the same type.	A way of combining predictions that belong to the different types.
2.	Aim to decrease variance, not bias.	Aim to decrease bias, not variance.
3.	Each model receives equal weight.	Models are weighted according to their performance.
4.	Each model is built independently.	New models are influenced by the performance of previously built models.
5.	Different training data subsets are selected using row sampling with replacement and random sampling methods from the entire training dataset.	Every new subset contains the elements that were misclassified by previous models.
6.	Bagging tries to solve the over-fitting problem.	Boosting tries to reduce bias.
7.	If the classifier is unstable (high variance), then apply bagging.	If the classifier is stable and simple (high bias) the apply boosting.
8.	In this base classifiers are trained parallelly.	In this base classifiers are trained sequentially.
9	Example: The Random forest model uses Bagging.	Example: The AdaBoost uses Boosting techniques

8. What is out-of-bag error in random forests?

Answer:

OOB (out-of-bag) errors are an estimate of the performance of a random forest classifier or regressor on unseen data. In scikit-learn, the OOB error can be obtained using the oob_score_ attribute of the random forest classifier or regressor.

The OOB error is computed using the samples that were not included in the training of the individual trees. This is different from the error computed using the usual training and validation sets, which are used to tune the hyperparameters of the random forest.

The OOB error can be useful for evaluating the performance of the random forest on unseen data. It is not always a reliable estimate of the generalization error of the model, but it can provide a useful indication of how well the model is performing.

To compute the OOB error, the samples that are not used in the training of an individual tree are known as “out-of-bag” samples. These samples are not used in the training of the tree, but they

are used to compute the OOB error for that tree. The OOB error for the entire random forest is computed by averaging the OOB errors of the individual trees.

One of the main use cases of the OOB error is to evaluate the performance of an ensemble model, such as a random forest. Because the OOB error is calculated using out-of-bag samples, which are samples that are not used in the training of the model, it provides an unbiased estimate of the model's performance.
Another use case of the OOB error is to tune the hyperparameters of a model. By using the OOB error as a performance metric, the hyperparameters of the model can be adjusted to improve its performance on unseen data.
Additionally, the OOB error can be used to diagnose whether a model is overfitting or underfitting. If the OOB error is significantly higher than the validation score, it may indicate that the model is overfitting and not generalizing well to unseen data. On the other hand, if the OOB error is significantly lower than the validation score, it may indicate that the model is underfitting and not learning the underlying patterns in the data.

9. What is K-fold cross-validation?

Answer:

k-fold cross-validation is one of the most popular strategies widely used by data scientists. It is a *data partitioning strategy* so that you can effectively use your dataset to build *a more generalized model*. The main intention of doing any kind of machine learning is to develop a more generalized model which can perform well on *unseen data*. One can build a perfect model on the training data with 100% accuracy or 0 error, but it may fail to generalize for unseen data. So, it is not a good model. It overfits the training data. Machine Learning is all about *generalization* meaning that model's performance can only be measured with data points that have never been used during the training process. That is why we often split our data into a training set and a test set.

Data splitting process can be done more effectively with k-fold cross-validation. Here, we discuss two scenarios which involve k-fold cross-validation. Both involve splitting the dataset, but with different approaches.

- Using k-fold cross-validation for evaluating a model's performance
- Using k-fold cross-validation for hyperparameter tuning

10. What is hyper parameter tuning in machine learning and why it is done?

Answer:

A Machine Learning model is defined as a mathematical model with several parameters that need to be learned from the data. By training a model with existing data, we can fit the model parameters.
However, there is another kind of parameter, known as *Hyperparameters*, that cannot be directly learned from the regular training process. They are usually fixed before the actual training process begins. These parameters express important properties of the model such as its complexity or how fast it should learn. This article aims to explore various strategies to tune hyperparameters for Machine learning models. In the context of machine learning, hyperparameters are configuration variables that are set before the training process of a model begins. They control the learning process itself, rather than being learned from the data. Hyperparameters are often used to tune the performance of a model, and they can have a significant impact on the model's accuracy, generalization, and other metrics.

The two best strategies for Hyperparameter tuning are:

1. GridSearchCV
    2. Grid search can be considered as a "brute force" approach to hyperparameter optimization. We fit the model using all possible combinations after creating a grid of potential discrete hyperparameter values. We log each set's model performance and then choose the combination that produces the best results. This approach is called GridSearchCV, because it searches for the best set of hyperparameters from a grid of hyperparameters values.
    3. An exhaustive approach that can identify the ideal hyperparameter combination is grid search. But the slowness is a disadvantage. It often takes a lot of processing power and time to fit the model with every potential combination, which might not be available.

2. RandomizedSearchCV
    3. As the name suggests, the random search method selects values at random as opposed to the grid search method's use of a predetermined set of numbers. Every iteration, random search attempts a different set of hyperparameters and logs the model's performance. It returns the combination that provided the best outcome after several iterations. This approach reduces unnecessary computation.
    4. RandomizedSearchCV solves the drawbacks of GridSearchCV, as it goes through only a fixed number of hyperparameter settings. It moves within the grid in a random fashion to find the best set of hyperparameters. The advantage is that, in most cases, a random search will produce a comparable result faster than a grid search.

3. Bayesian Optimization

Grid search and random search are often inefficient because they evaluate many unsuitable hyperparameter combinations without considering the previous iterations' results. Bayesian optimization, on the other hand, treats the search for optimal hyperparameters as an optimization problem. It considers the previous evaluation results when selecting the next hyperparameter combination and applies a probabilistic function to choose the combination that will likely yield

the best results. This method discovers a good hyperparameter combination in relatively few iteration

11. What issues can occur if we have a large learning rate in Gradient Descent?

Answer:

The learning rate is an important hyperparameter that greatly affects the performance of gradient descent. It determines how quickly or slowly our model learns, and it plays an important role in controlling both convergence and divergence of the algorithm. When the learning rate is too large, gradient descent can suffer from divergence. This means that weights increase exponentially, resulting in exploding gradients which can cause problems such as instabilities and overly high loss values. On the other hand, if the learning rate is too small, then gradient descent can suffer from slow convergence or even stagnation—which means it may not reach a local minimum at all unless many iterations are performed on large datasets.

In order to avoid these issues with different learning rates for each parameter/variable, we use adaptive techniques such as Adagrad and Adam which adjust their own learning rates throughout training based on real-time observations of parameters during optimization (i.e., they control exploration/exploitation trade-offs). These adaptive measures ensure better results than standard gradient descent while avoiding potential pitfalls in terms of either massive gains or slow losses due to misconfigured static global learning rates like those used with traditional gradient descent algorithms.

12. Can we use Logistic Regression for classification of Non-Linear Data? If not, why?

Answer:

Non-linear data is data that cannot be separated by a straight line. For example, if you have two classes of data points that form a circle and a square, you cannot draw a line that divides them perfectly. Non-linear data is common in real-world problems, such as image recognition, natural language processing, and fraud detection.

Logistic regression is a type of linear model that predicts the probability of a binary outcome, such as yes or no, true or false, or 0 or 1. It uses a logistic function, also known as a sigmoid function, to map the input features to a value between 0 and 1. The logistic function has an S-shaped curve that flattens at the extremes. Logistic regression then uses a threshold, usually 0.5, to classify the output as 0 or 1.

Even though the idea behind the generation of non linear dataset is pretty simple, there were plenty

of data points, and no noise at all, the logistic regression model performed poorly on non linear

dataset.

The reason is that the target label has no linear correlation with the features. In such cases, logistic regression (or linear regression for regression problems) can't predict targets with good accuracy (even on the training data).

13. Differentiate between Adaboost and Gradient Boosting.

Answer:

Loss Function:

The technique of Boosting uses various loss functions. In case of Adaptive Boosting or AdaBoost, it minimises the exponential loss function that can make the algorithm sensitive to the outliers. With Gradient Boosting, any differentiable loss function can be utilised. Gradient Boosting algorithm is more robust to outliers than AdaBoost.

Flexibility

AdaBoost is the first designed boosting algorithm with a particular loss function. On the other hand, Gradient Boosting is a generic algorithm that assists in searching the approximate solutions to the additive modelling problem. This makes Gradient Boosting more flexible than AdaBoost.

Benefits

AdaBoost minimises loss function related to any classification error and is best used with weak learners. The method was mainly designed for binary classification problems and can be utilised to boost the performance of decision trees. Gradient Boosting is used to solve the differentiable loss function problem. The technique can be used for both classification and regression problems.

Shortcomings

In the case of Gradient Boosting, the shortcomings of the existing weak learners can be identified by gradients and with AdaBoost, it can be identified by high-weight data points.

Wrapping Up

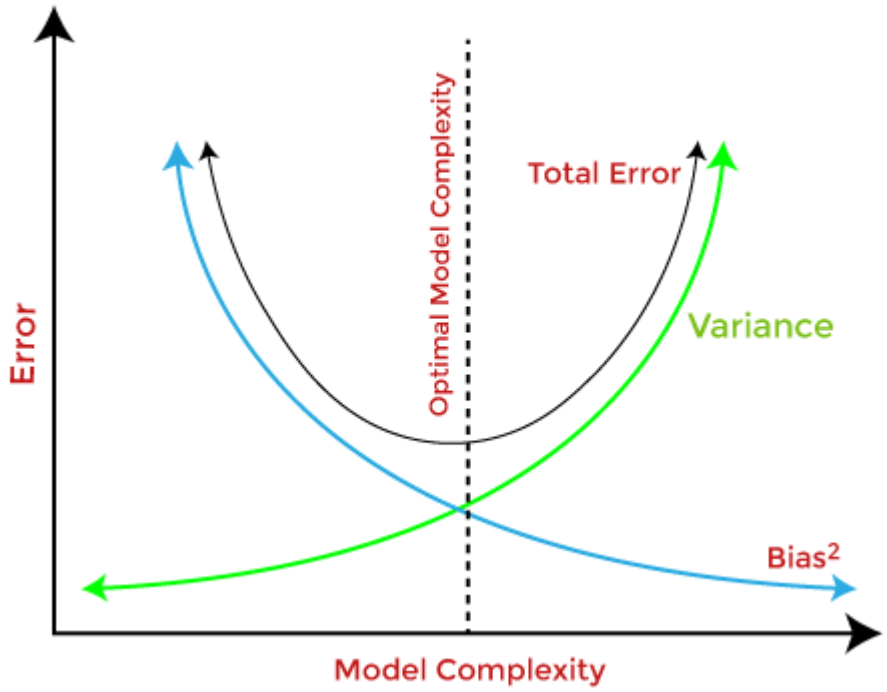
Though there are several differences between the two boosting methods, both the algorithms follow the same path and share similar historic roots. Both the algorithms work for boosting the performance of a simple base-learner by iteratively shifting the focus towards problematic observations that are challenging to predict.

In the case of AdaBoost, the shifting is done by up-weighting observations that were misclassified before, while Gradient Boosting identifies the difficult observations by large residuals computed in the previous iterations.

14. What is bias-variance trade off in machine learning?

Answer:

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 supervised 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.***

15. Give short description each of Linear, RBF, Polynomial kernels used in SVM

Answer:

In Support Vector Machines (SVMs), there are several types of kernel functions that can be used to map the input data into a higher-dimensional feature space. The choice of kernel function depends on the specific problem and the characteristics of the data.

Here are some most commonly used kernel functions in SVMs:

Linear Kernel

A linear kernel is a type of kernel function used in machine learning, including in SVMs (Support Vector Machines). It is the simplest and most commonly used kernel function, and it defines the dot product between the input vectors in the original feature space.

The linear kernel can be defined as:

1. $K(x, y) = x .y$

Where x and y are the input feature vectors. The dot product of the input vectors is a measure of their similarity or distance in the original feature space.

When using a linear kernel in an SVM, the decision boundary is a linear hyperplane that separates the different classes in the feature space. This linear boundary can be useful when the data is already separable by a linear decision boundary or when dealing with high-dimensional data, where the use of more complex kernel functions may lead to overfitting.

Polynomial Kernel

A particular kind of kernel function utilised in machine learning, such as in SVMs, is a polynomial kernel (Support Vector Machines). It is a nonlinear kernel function that employs polynomial functions to transfer the input data into a higher-dimensional feature space.

One definition of the polynomial kernel is:

Where x and y are the input feature vectors, c is a constant term, and d is the degree of the polynomial, K(x, y) = (x. y + c)d. The constant term is added to, and the dot product of the input vectors elevated to the degree of the polynomial.

The decision boundary of an SVM with a polynomial kernel might capture more intricate correlations between the input characteristics because it is a nonlinear hyperplane.

The degree of nonlinearity in the decision boundary is determined by the degree of the polynomial.

The polynomial kernel has the benefit of being able to detect both linear and nonlinear correlations in the data. It can be difficult to select the proper degree of the polynomial, though, as a larger degree

can result in overfitting while a lower degree cannot adequately represent the underlying relationships in the data.

In general, the polynomial kernel is an effective tool for converting the input data into a higher-dimensional feature space in order to capture nonlinear correlations between the input characteristics.

Gaussian (RBF) Kernel

The Gaussian kernel, also known as the radial basis function (RBF) kernel, is a popular kernel function used in machine learning, particularly in SVMs (Support Vector Machines). It is a nonlinear kernel function that maps the input data into a higher-dimensional feature space using a Gaussian function.

The Gaussian kernel can be defined as:

1. K(x, y) = exp(-gamma * ||x - y||^2)

Where x and y are the input feature vectors, gamma is a parameter that controls the width of the Gaussian function, and ||x - y||^2 is the squared Euclidean distance between the input vectors.

When using a Gaussian kernel in an SVM, the decision boundary is a nonlinear hyper plane that can capture complex nonlinear relationships between the input features. The width of the Gaussian function, controlled by the gamma parameter, determines the degree of nonlinearity in the decision boundary.

One advantage of the Gaussian kernel is its ability to capture complex relationships in the data without the need for explicit feature engineering. However, the choice of the gamma parameter can be challenging, as a smaller value may result in under fitting, while a larger value may result in over fitting.

Laplace Kernel

The Laplacian kernel, also known as the Laplace kernel or the exponential kernel, is a type of kernel function used in machine learning, including in SVMs (Support Vector Machines). It is a non-parametric kernel that can be used to measure the similarity or distance between two input feature vectors.

The Laplacian kernel can be defined as:

1. K(x, y) = exp(-gamma * ||x - y||)

Where x and y are the input feature vectors, gamma is a parameter that controls the width of the Laplacian function, and ||x - y|| is the L1 norm or Manhattan distance between the input vectors.

When using a Laplacian kernel in an SVM, the decision boundary is a nonlinear hyperplane that can capture complex relationships between the input features. The width of the Laplacian function, controlled by the gamma parameter, determines the degree of nonlinearity in the decision boundary.

One advantage of the Laplacian kernel is its robustness to outliers, as it places less weight on large distances between the input vectors than the Gaussian kernel. However, like the Gaussian kernel, choosing the correct value of the gamma parameter can be challenging.