MACHINE LEARNING ASSIGNMENT - 5 ANSWER

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?

Ans- **R-square** is a goodness-of-fit measure for linear regression models. This statistic indicates the percentage of the variance in the dependent variable that the independent variables explain collectively. R-squared measures the strength of the relationship between your model and the dependent variable on a convenient 0 – 100% scale.

After fitting a linear regression model, you need to determine how well the model fits the data. Does it do a good job of explaining changes in the dependent variable? There are several key goodness-of-fit statistics for regression analysis.

The **residual sum of squares (RSS)** measures the level of variance in the error term, or residuals, of a regression model. The smaller the residual sum of squares, the better your model fits your data; the greater the residual sum of squares, the poorer your model fits your data.
The residual sum of squares (RSS) is the sum of the squared distances between your actual versus your predicted values:

RSS=∑i=1n(yi− ŷi)2

Where yi is a given datapoint and ŷi is your fitted value for yi.

R2 is a good metric for linear regression.

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.

Ans - **Total sum of squares:** TSS represents the total sum of squares. It is the squared values of the dependent variable to the sample mean. In other words, the total sum of squares measures the variation in a sample. Sum of squares regression: Sum of squares due to regression is represented by SSR.

$$\mathrm{TSS} = \sum_{i=1}^{n}(y_i - \bar{y})^2$$

TSS	=	total sum of squares
n	=	number of observations
yi	=	value in a sample
ȳ	=	mean value of a sample

The **explained sum of squares (ESS**) or **Regression sum of squares,** alternatively known as the **model sum of squares** or **sum of squares due to regression (SSR)** is the sum of the squares of the deviations of the predicted values from the mean value of a response variable, in a standard regression model.

$$\mathrm{SSR} = \sum_{\mathrm{i}=1}^{\mathrm{n}}(\hat{\mathrm{y}}_\mathrm{i} - \bar{\mathrm{y}})^2$$

Where:
ŷi = the value estimated by the regression line
ȳ = the mean value of a sample

The **residual sum of squares (RSS)** measures the level of variance in the error term, or residuals, of a regression model. The smaller the residual sum of squares, the better your model fits your data; the greater the residual sum of squares, the poorer your model fits your data.
The residual sum of squares (RSS) is the sum of the squared distances between your actual versus your predicted values:

$$\mathrm{SSE} = \sum_{\mathrm{i}=1}^{\mathrm{n}}(\mathrm{y}_\mathrm{i} - \hat{\mathrm{y}}_\mathrm{i})^2$$

Where yi is a given datapoint and ŷi is your fitted value for yi or the value estimated by the regression line.

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

Ans- Regularization refers to techniques used to calibrate machine learning models to minimize the adjusted loss function and prevent overfitting or underfitting. Using Regularization, we can fit our machine learning model appropriately on a given test set and hence reduce the errors in it.

Regularization works by adding a penalty or complexity term to the complex model. Let's consider the simple linear regression equation:

y= β0+β1x1+β2x2+β3x3+⋯+βnxn +b

In the above equation, Y represents the value to be predicted

X1, X2, …Xn are the features for Y.

β0,β1,…..βn are the weights or magnitude attached to the features, respectively. Here represents the bias of the model, and b represents the intercept.

Now, we will add a loss function and optimize parameters to make the model that can predict the accurate value of Y. The loss function for the linear regression is called as RSS or Residual sum of squares.

There are mainly two types of regularization techniques, which are given below:

Ridge Regression

Lasso Regression

4. What is Gini–impurity index?

Ans- Gini Index, also known as Gini impurity, calculates the amount of probability of a specific feature that is classified incorrectly when selected randomly. If all the elements are linked with a single class then it can be called pure.

Let's perceive the criterion of the Gini Index, like the properties of entropy, the Gini index varies between values 0 and 1, where 0 expresses the purity of classification, i.e. All the elements belong to a specified class or only one class exists there. And 1 indicates the random distribution of elements across various classes. The value of 0.5 of the Gini Index shows an equal distribution of elements over some classes.

Consider dataset *D* that contains samples from *k* classes. The probability of samples belonging to class *i* at a given node can be denoted as *pi*. Then the Gini Impurity of *D* is defined as:

$$Gini(D) = 1 - \sum_{i=1}^{k} p_i^2$$

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

Ans- 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 a smaller sample of events that meet the previous assumptions. This small sample could lead to unsound conclusions.

An example of this could be predicted 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. Does the third level ask 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 tonight's 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 is playing. This is a simpler model with less variance from sample to sample but ultimately will not be a strong predictive model.

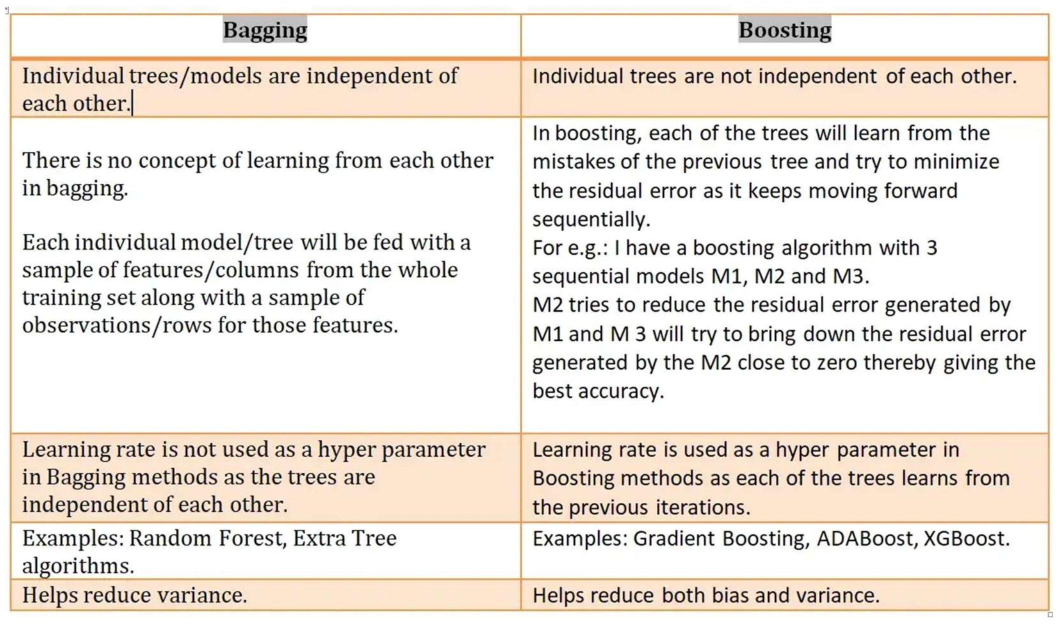
Ideally, we would like to minimize both errors due to bias and errors due to variance. Enter random forests. Random forests mitigate this problem well. A random forest is simply a collection of decision trees whose results are aggregated into one final result. Their ability to limit overfitting without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

6. What is an ensemble technique in machine learning?

Ans- One of the most effective machine learning methodologies is ensemble modeling, also known as ensembles. Ensemble modeling combines statistical techniques to create a model that produces a unified prediction. It is through combining estimates and following the wisdom of the crowd that ensemble modeling performs a final classification or outcome with better predictive performance.
An ensemble method is a technique which uses multiple independent similar or different models/weak learners to derive an output or make some predictions.
For e.g.
A random forest is an ensemble of multiple decision trees.
An ensemble can also be built with a combination of different models like random forest, SVM, Logistic regression etc.

Types of ensemble methods
There are many ensemble techniques available but we will discuss about the below two most widely used methods:
1. Bagging
2. Boosting

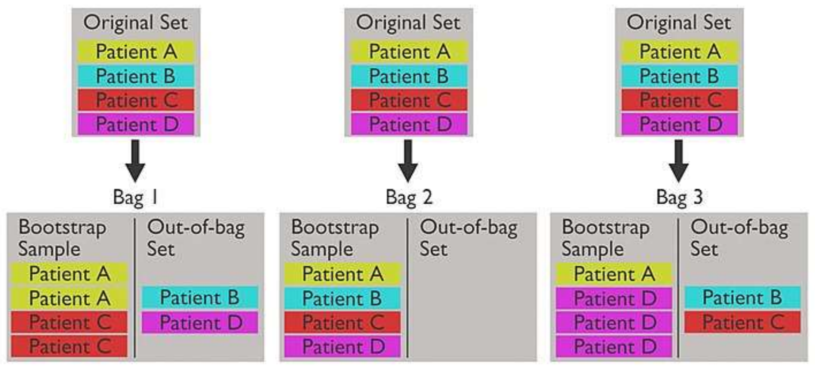
7. What is the difference between Bagging and Boosting techniques?



Bagging	**Boosting**
Individual trees/models are independent of each other.	Individual trees are not independent of each other.
There is no concept of learning from each other in bagging. Each individual model/tree will be fed with a sample of features/columns from the whole training set along with a sample of observations/rows for those features.	In boosting, each of the trees will learn from the mistakes of the previous tree and try to minimize the residual error as it keeps moving forward sequentially. For e.g.: I have a boosting algorithm with 3 sequential models M1, M2 and M3. M2 tries to reduce the residual error generated by M1 and M 3 will try to bring down the residual error generated by the M2 close to zero thereby giving the best accuracy.
Learning rate is not used as a hyper parameter in Bagging methods as the trees are independent of each other.	Learning rate is used as a hyper parameter in Boosting methods as each of the trees learns from the previous iterations.
Examples: Random Forest, Extra Tree algorithms.	Examples: Gradient Boosting, ADABoost, XGBoost.
Helps reduce variance.	Helps reduce both bias and variance.

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

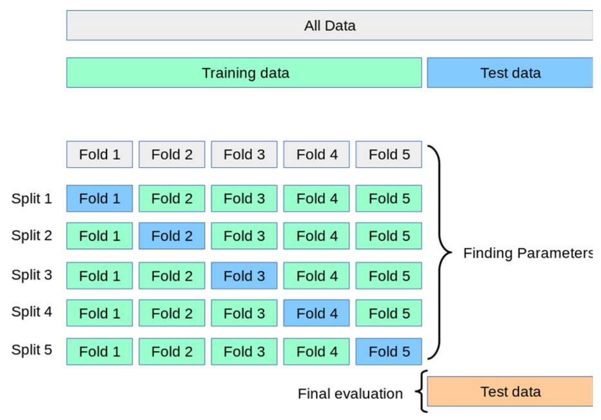
Ans- The out-of-bag (OOB) error is the average error for each calculated using predictions from the trees that do not contain in their respective bootstrap sample. This allows the Random Forest Classifier to be fit and validated whilst being trained.



This example shows how bagging could be used in the context of diagnosing disease. A set of patients are the original dataset, but each model is trained only by the patients in its bag. The patients in each out-of-bag set can be used to test their respective models. The test would consider whether the model can accurately determine if the patient has the disease. Visualizing the bagging process. Sampling 4 patients from the original set with replacement and showing the out-of-bag sets. Only patients in the bootstrap sample would be used to train the model for that bag.

9. What is K-fold cross-validation?

Ans - 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

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

The general process of k-fold cross-validation for evaluating a model's performance is:

- The whole dataset is randomly split into independent k-folds without replacement.
- k-1 folds are used for the model training and one fold is used for performance evaluation.
- This procedure is repeated k times (iterations) so that we obtain k number of performance estimates (e.g. MSE) for each iteration.
- Then we get the mean of k number of performance estimates (e.g. MSE).
- Remark 1: The splitting process is done without replacement. So, each observation will be used for training and validation exactly once.

- Remark 2: Good standard values for k in k-fold cross-validation are 5 and 10. However, the value of k depends on the size of the dataset. For small datasets, we can use higher values for k. However, larger values of k will also increase the runtime of the cross-validation algorithm and the computational cost.
- Remark 3: When k=5, 20% of the test set is held back each time. When k=10, 10% of the test set is held back each time and so on…
- Remark 4: A special case of k-fold cross-validation is the Leave-one-out cross-validation (LOOCV) method in which we set k=n (number of observations in the dataset). Only one training sample is used for testing during each iteration. This method is very useful when working with very small datasets.

Using k-fold cross-validation for hyperparameter tuning

Model parameters are the parameters which learn during the training process. We do not manually set values for the parameters and they learn from the data that we provide. In contrast, the model hyperparameters are the parameters that do not learn from data. So, we have to set values for them manually. We always set values for the model hyperparameters at the creation of a particular model and before we start the training process. Grid search is a popular hyperparameter optimization technique. It is looking for an optimal combination of hyperparameter values in a way that it can further improve the performance of a model. Using k-fold cross-validation in combination with grid search is a very useful strategy to improve the performance of a machine learning model by tuning the model hyperparameters. In grid search, we can set values for multiple hyperparameters. Let's say we have two hyperparameters each having three different values. So, we have 9 (3 x 3) different combinations of hyperparameter values. The space in which all those combinations contain is called the hyperparameter space. When there are two hyperparameters, space is two-dimensional.
In grid search, the algorithm takes one combination of hyperparameter values at a time from the hyperparameter space that we have specified. Then it trains the model using those hyperparameter values and evaluates it through k-fold cross-validation. It stores the performance estimate (e.g. MSE). Then the algorithm takes another combination of hyperparameter values and does the same. After taking all the combinations, the algorithm stores all the performance estimates. Out of those estimates, it selects the best one. The combination of hyperparameter values that yields the best performance estimate is the optimal combination of hyperparameter values. The best model includes those hyperparameter values.
The reason behind fitting the best model to the whole training set after k-fold cross-validation is to provide more training samples to the learning algorithm of the best model. This usually results in a more accurate and robust model.

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

Ans- A Machine Learning model is defined as a mathematical model with a number of parameters that need to be learned from the data. By training a model with existing data, we are able to 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.

Some examples of model hyperparameters include:

- The penalty in Logistic Regression Classifier i.e. L1 or L2 regularization
- The learning rate for training a neural network.
- The C and sigma hyperparameters for support vector machines.
- The k in k-nearest neighbors.
- The aim of this article is to explore various strategies to tune hyperparameters for Machine learning models.

Models can have many hyperparameters and finding the best combination of parameters can be treated as a search problem. The two best strategies for Hyperparameter tuning are:

- GridSearchCV
- RandomizedSearchCV

In GridSearchCV approach, the machine learning model is evaluated for a range of hyperparameter values. This approach is called GridSearchCV, because it searches for the best set of hyperparameters from a grid of hyperparameters values.

For example, if we want to set two hyperparameters C and Alpha of the Logistic Regression Classifier model, with different sets of values. The grid search technique will construct many versions of the model with all possible combinations of hyperparameters and will return the best one.

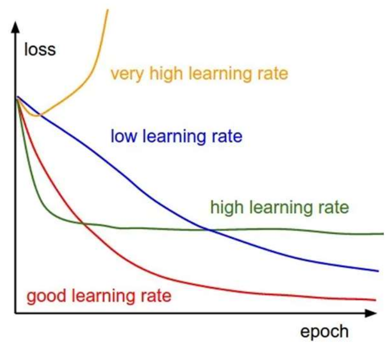
As in the image, for C = [0.1, 0.2, 0.3, 0.4, 0.5] and Alpha = [0.1, 0.2, 0.3, 0.4]. For a combination of C=0.3 and Alpha=0.2, the performance score comes out to be 0.726(Highest), therefore it is selected.
Drawback: GridSearchCV will go through all the intermediate combinations of hyperparameters which makes grid search computationally very expensive.

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. This approach reduces unnecessary computation.

Why hyperparameter is important:
Hyperparameter tuning takes advantage of the processing to test different hyperparameter configurations when training your model. It can give you optimized values for hyperparameters, which maximizes your model's predictive accuracy.

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

Ans- The learning rate is the most important hyper-parameter — there is a gigantic
amount of material on how to choose a learning rate, how to modify the learning rate during the training, and how the wrong learning rate can completely ruin the model training.
Maybe you've seen this famous graph below that shows how a learning rate that is too big or too small affects the loss during training.



Suppose, you are coming back from hiking in the mountains and you want to get back home as quickly as possible. At some point in your path, you can either choose to go ahead or make a right turn.
The path ahead is almost flat, while the path to your right is kind of steep. The
steepness is the gradient. If you take a single step one way or the other, it will lead to different outcomes (you'll descend more if you take one step to the right instead of going ahead).
But, here is the thing: you know that the path to your right is getting you home faster, so you don't take just one step, but multiple steps in that direction: the steeper the path, the more steps you take! Remember, "more impact, more steps"! You just cannot resist the urge to take that many steps; your behavior seems to be completely determined by the landscape.
But, you still have one choice: you can adjust the size of your step. You can choose to take steps of any size, from tiny steps to long strides. That's your learning rate.

updated location = previous location + step size * number of steps
Now, compare it to what we did with the parameters:
updated value = previous value - learning rate * gradient

At this point, after moving in one direction (say, the right turn), you'd have to stop and move in the other direction (for just a fraction of a step, because the path was almost flat). And so on and so forth… Well, I don't think anyone has ever returned from hiking in such an orthogonal zigzag path…
Anyway, let's explore further the only choice you have: the size of your step, I
mean, the learning rate. Choose your learning rate wisely.

Using a small learning rate. Small learning rates are safe, as expected. If you were to take tiny steps while returning home from your hiking, you'd be more likely to arrive there safe and sound — but it would take a lot of time. The same holds true for training models: small learning rates will likely get you to (some) minimum point, eventually. Unfortunately, time is money, especially when you're paying for GPU time in the cloud… so, there is an incentive to try bigger learning rates.

If the learning rate is very large we will skip the optimal solution. If it is too small we will need too many iterations to converge to the best values. Increasing the learning rate further will cause an increase in the loss as the parameter updates to cause the loss to "bounce around" and even diverge from the minima. So using a good learning rate is crucial.

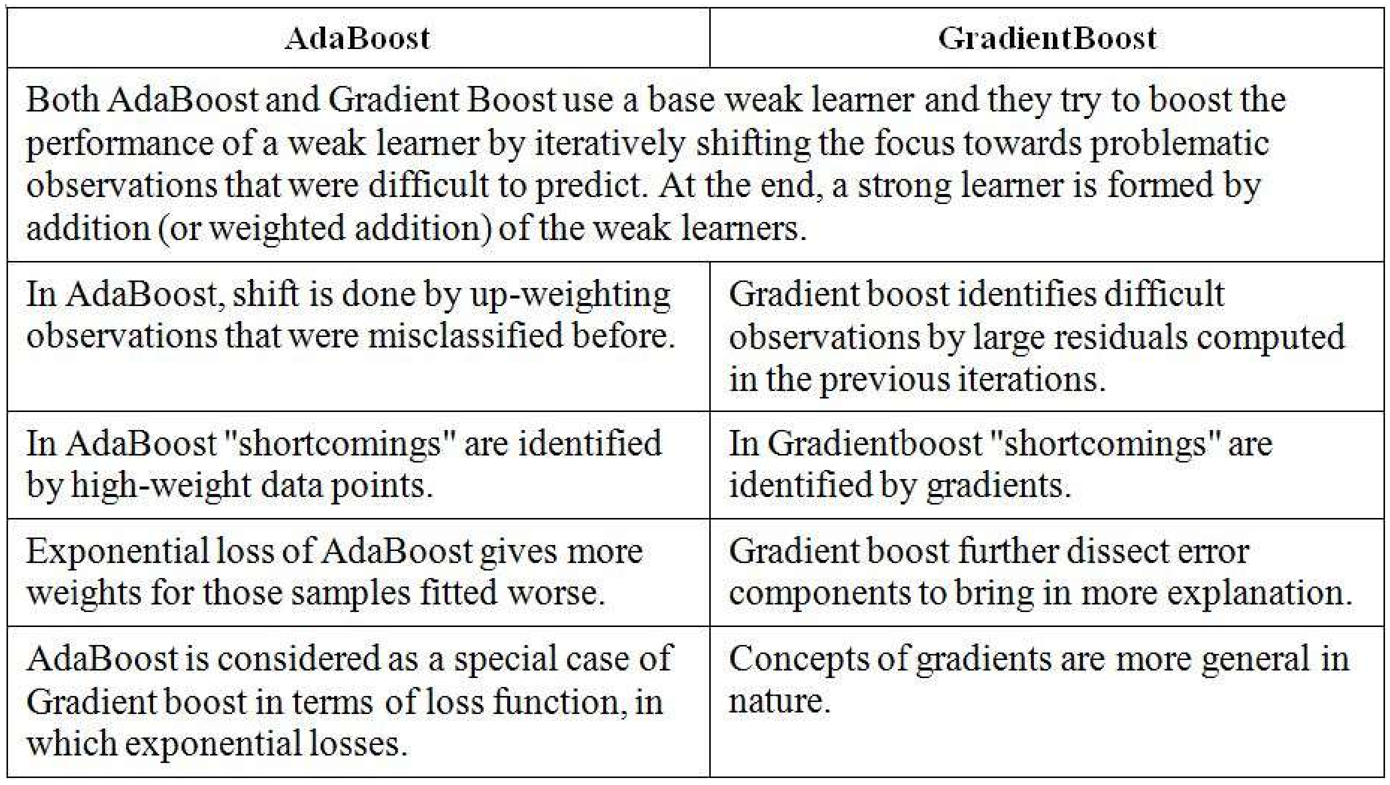
12. Can we use Logistic Regression for classification of Non-Linear Data? If not, why?

Ans- Non-linear problems can't be solved with logistic regression because it has a linear decision surface. Linearly separable data is rarely found in real-world scenarios.

Logistic regression has traditionally been used to come up with a hyperplane that separates the feature space into classes. But if we suspect that the decision boundary is non-linear we may get better results by attempting some nonlinear functional forms for the logit function.

13. Differentiate between Adaboost and Gradient Boosting.

Ans-



AdaBoost	GradientBoost
Both AdaBoost and Gradient Boost use a base weak learner and they try to boost the performance of a weak learner by iteratively shifting the focus towards problematic observations that were difficult to predict. At the end, a strong learner is formed by addition (or weighted addition) of the weak learners.	
In AdaBoost, shift is done by up-weighting observations that were misclassified before.	Gradient boost identifies difficult observations by large residuals computed in the previous iterations.
In AdaBoost "shortcomings" are identified by high-weight data points.	In Gradientboost "shortcomings" are identified by gradients.
Exponential loss of AdaBoost gives more weights for those samples fitted worse.	Gradient boost further dissect error components to bring in more explanation.
AdaBoost is considered as a special case of Gradient boost in terms of loss function, in which exponential losses.	Concepts of gradients are more general in nature.

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

Ans- In supervised machine learning an algorithm learns a model from training data. The goal of any supervised machine learning algorithm is to best estimate the mapping function (f) for the output variable (Y) given the input data (X). The mapping function is often called the target function because it is the function that a given supervised machine learning algorithm aims to approximate.

Bias-Variance Trade-Off

The goal of any supervised machine learning algorithm is to achieve low bias and low variance. In turn the algorithm should achieve good prediction performance.

You can see a general trend in the examples above:

- Linear machine learning algorithms often have a high bias but a low variance.
- Nonlinear machine learning algorithms often have a low bias but a high variance.

The parameterization of machine learning algorithms is often a battle to balance out bias and variance.

Below are two examples of configuring the bias-variance trade-off for specific algorithms:

- The k-nearest neighbors algorithm has low bias and high variance, but the trade-off can be changed by increasing the value of k which increases the number of neighbors that contribute t the prediction and in turn increases the bias of the model.
- The support vector machine algorithm has low bias and high variance, but the trade-off can be changed by increasing the C parameter that influences the number of violations of the margin allowed in the training data which increases the bias but decreases the variance.

There is no escaping the relationship between bias and variance in machine learning.

- Increasing the bias will decrease the variance.
- Increasing the variance will decrease the bias.

There is a trade-off at play between these two concerns and the algorithms you choose and the way you choose to configure them are finding different balances in this trade-off for your problem

In reality, we cannot calculate the real bias and variance error terms because we do not know the actual underlying target function. Nevertheless, as a framework, bias and variance provide the tools to understand the behaviour of machine learning algorithms in the pursuit of predictive performance.

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

Ans- **SVM** is a famous supervised machine learning algorithm used for classification as well as regression algorithms. However, mostly it is preferred for classification algorithms. It basically separates different target classes in a hyperplane in n-dimensional or multidimensional space.

The main motive of the SVM is to create the best decision boundary that can separate two or more classes(with maximum margin) so that we can correctly put new data points in the correct class.
It chooses extreme vectors or support vectors to create the hyperplane.

SVM Kernel Functions

SVM algorithms use a group of mathematical functions that are known as kernels. The function of a kernel is to require data as input and transform it into the desired form.

Different SVM algorithms use differing kinds of kernel functions. These functions are of different kinds—for instance, linear, nonlinear, polynomial, radial basis function (RBF), and sigmoid.

The most preferred kind of kernel function is RBF. Because it's localized and has a finite response along the complete x-axis.

The kernel functions return the scalar product between two points in an exceedingly suitable feature space. Thus by defining a notion of resemblance, with a little computing cost even in the case of very high-dimensional spaces.

$$K\left(\bar{x}\right) = \begin{matrix} 1 & \text{if}\left\|\bar{x}\right\| \le 1 \\ 0 & \text{otherwise} \end{matrix}$$

Linear Kernel

It is the most basic type of kernel, usually one dimensional in nature. It proves to be the best function when there are lots of features. The linear kernel is mostly preferred for text-classification problems as most of these kinds of classification problems can be linearly separated.

Linear kernel functions are faster than other functions.

Linear Kernel Formula

F(x, xj) = sum(x.xj)

Here, x, xj represents the data you're trying to classify.

Polynomial Kernel

It is a more generalized representation of the linear kernel. It is not as preferred as other kernel functions as it is less efficient and accurate.

Polynomial Kernel Formula

F(x, xj) = (x.xj+1)^d

Here '.' shows the dot product of both the values, and d denotes the degree.

F(x, xj) representing the decision boundary to separate the given classes.

Gaussian Radial Basis Function (RBF)

It is one of the most preferred and used kernel functions in svm. It is usually chosen for non-linear data. It helps to make proper separation when there is no prior knowledge of data.

Gaussian Radial Basis Formula

$$F(x, xj) = \exp(-\text{gamma} * ||x - xj||^2)$$

The value of gamma varies from 0 to 1. You have to manually provide the value of gamma in the code. The most preferred value for gamma is 0.1.