1. **What is the purpose of the General Linear Model (GLM)**

The generalized linear model (GLM) generalizes linear regression by allowing the linear model to be related to the response variable via a link function and allowing the magnitude of the variance of each measurement to be a function of its predicted value.

**2. What are the key assumptions of the General Linear Mode**

Data should be independent and random (Each Random variable has the same probability distribution).

* The response variable ydoes not need to be normally distributed, but the distribution is from an exponential family (e.g. binomial, Poisson, multinomial, normal)
* The original response variable need not have a linear relationship with the independent variables, but the transformed response variable (through the link function) is linearly dependent on the independent variables

Ex., Logistic Regression Equation, Log odds = β0+β1X1+β2X2 ,

where β0,β1,β2 are regression coefficient, and X1,X2 are the independent variables

* Feature engineering on the Independent variable can be applied i.e instead of taking the original raw independent variables, variable transformation can be done, and the transformed independent variables, such as taking a log transformation, squaring the variables, reciprocal of the variables, can also be used to build the GLM model.
* Homoscedasticity (i.e constant variance) need not be satisfied. Response variable Error variance can increase, or decrease with the independent variables.
* Errors are independent but need not be normally distributed

**3. How do you interpret the coefficients in a GLM?**

The GLM coefficients only show the multiplicative change in odds ratio. so if p1 is the risk of getting a high score for black defendants and p0 is the risk of getting a high score for white defendants, then exp(0.47721) shows (p1/(1-p1))/(p0/(1-p0))

**4. What is the difference between a univariate and multivariate GLM?**

The most basic difference is that univariate regression has one explanatory (predictor) variable

x

and multivariate regression has more at least two explanatory (predictor) variables

x

1

,

x

2

,

...

,

x

n

. In both situations there is one response variable

y

. Let me know if you want more detai**l.**

**5. Explain the concept of interaction effects in a GLM.**

Interaction effects in GLMs describing probabilities and counts are not equal to product terms between predictor variables. Instead, interactions may be functions of the predictors of a model, requiring nontraditional approaches for interpreting these effects accurately

**7. What is the purpose of the design matrix in a GLM?**

The design matrix is used in certain statistical models, e.g., the general linear model. It can contain indicator variables (ones and zeros) that indicate group membership in an ANOVA, or it can contain values of continuous variables.

**9.What is the difference between Type I, Type II, and Type III sums of squares in a GLM?**

Changing this will not affect Type I or Type II tests.

Type I sum of squares are “sequential.” In essence the factors are tested in the order they are listed in the model.

Type III sum of squares are “partial.” In essence, every term in the model is tested in light of every other term in the model. That means that main effects are tested in light of interaction terms as well as in light of other main effects.

Type II sum of squares are similar to Type III, except that they preserve the principle of marginality. This means that main factors are tested in light of one another, but not in light of the interaction term.

When data are balanced and the design is simple, types I, II, and III will give the same results. But readers should be aware that results may differ for unbalanced data or more complex designs. The code below gives an example of this.

There are disagreements as to which type should be used routinely in analysis of variance. In reality, the user should understand what hypothesis she wants to test, and then choose the appropriate tests. As general advice, I would recommend not using Type I except in cases where you intend to have the effects assessed sequentially. Beyond that, probably a majority of those in the R community recommend Type II tests, while SAS users are more likely to consider Type III tests.

**10 . Explain the concept of deviance in a GLM**

Deviance is a measure of error; lower deviance means better fit to data.

The greater the deviance, the worse the model fits compared to the best case (saturated).

Deviance is a quality-of-fit statistic for a model that is often used for statistical hypothesis testing.

**11. What is regression analysis and what is its purpose?**

Regression analysis is a statistical method that shows the relationship between two or more variables. Usually expressed in a graph, the method tests the relationship between a dependent variable against independent variables. Typically, the independent variable(s) changes with the dependent variable(s) and the regression analysis attempts to answer which factors matter most to that change.

**12. What is the difference between simple linear regression and multiple linear regression?**

Simple linear regression has only one x and one y variable.

Multiple linear regression has one y and two or more x variables.

For instance, when we predict rent based on square feet alone that is simple linear regression.

**13. How do you interpret the R-squared value in regression?**

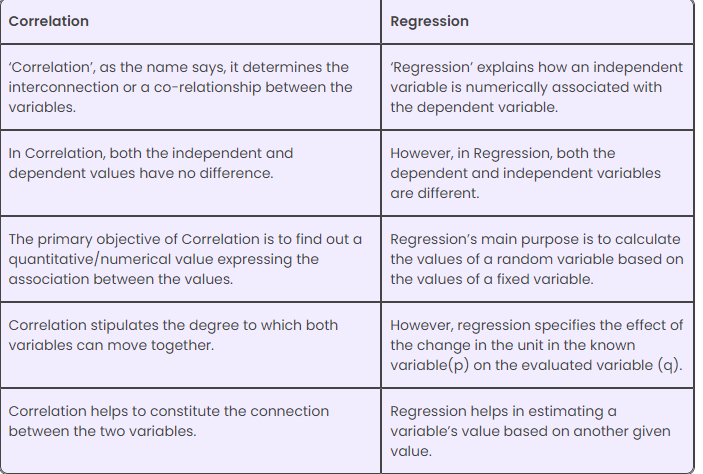
R-squared (R2) is a statistical measure that represents the proportion of the variance for a dependent variable that’s explained by an independent variable in a regression model.

R-squared is a statistical measure that indicates how much of the variation of a dependent variable is explained by an independent variable in a regression model.

In investing, R-squared is generally interpreted as the percentage of a fund’s or security’s price movements that can be explained by movements in a benchmark index.

An R-squared of 100% means that all movements of a security (or other dependent variable) are completely explained by movements in the index (or whatever independent variable you are interested in).

**14. What is the difference between correlation and regression?**

****

**15. What is the difference between the coefficients and the intercept in regression?**

The simple linear regression model is essentially a linear equation of the form y = c + b\*x; where y is the dependent variable (outcome), x is the independent variable (predictor), b is the slope of the line; also known as regression coefficient and c is the intercept; labeled as constant.

**16. How do you handle outliers in regression analysis?**

One option to dealing with outliers can be to drop the observations altogether. This can be a suitable option if it can be determined through further investigation that the survey entry was made in error. Perhaps a search of property tax records or even personal knowledge of properties in the neighborhood can lead to this conclusion.

Replacing The Outlier With a Another Value

**17. What is the difference between ridge regression and ordinary least squares regression?**

In summary, when there is a difference in variance between predictor variables, OLS tends to give higher variance for coefficients corresponding to predictors with higher variance, while Ridge Regression reduces the variance differences between coefficients by shrinking them towards zero.

18**. What is heteroscedasticity in regression and how does it affect the model?**

Heteroscedasticity means unequal scatter. In regression analysis, we talk about heteroscedasticity in the context of the residuals or error term. Specifically, heteroscedasticity is a systematic change in the spread of the residuals over the range of measured values. Heteroscedasticity is a problem because ordinary least squares (OLS) regression assumes that all residuals are drawn from a population that has a constant variance (homoscedasticity).

To satisfy the regression assumptions and be able to trust the results, the residuals should have a constant variance. In this blog post, I show you how to identify heteroscedasticity, explain what produces it, the problems it causes, and work through an example to show you several solutions.

If heteroskedasticity exists, the population used in the regression contains unequal variance, the analysis results may be invalid.

**20. What is polynomial regression and when is it used?**

A polynomial regression model is a machine learning model that can capture non-linear relationships between variables by fitting a non-linear regression line, which may not be possible with simple linear regression. It is used when linear regression models may not adequately capture the complexity of the relationship.

**21. What is a loss function and what is its purpose in machine learning?**

A loss function is a measure of how good your prediction model does in terms of being able to predict the expected outcome (or value). We convert the learning problem into an optimization problem, define a loss function and then optimize the algorithm to minimize the loss function.

**22. What is the difference between a convex and non-convex loss function?**

A convex function is one in which a line drawn between any two points on the graph lies on the graph or above it. There is only one requirement.

A non-convex function is one in which a line drawn between any two points on the graph may cross additional points. It was described as “wavy.”

**23. What is mean squared error (MSE) and how is it calculated?**

The Mean Squared Error measures how close a regression line is to a set of data points. It is a risk function corresponding to the expected value of the squared error loss.

Mean square error is calculated by taking the average, specifically the mean, of errors squared from data as it relates to a function

**24. What is mean absolute error (MAE) and how is it calculated?**

Mean Absolute Error calculates the average difference between the calculated values and actual values. It is also known as scale-dependent accuracy as it calculates error in observations taken on the same scale. It is used as evaluation metrics for regression models in machine learning

**25. What is log loss (cross-entropy loss) and how is it calculated?**

Binary Cross Entropy is the negative average of the log of corrected predicted probabilities. Binary Cross Entropy, also known as Binary Log Loss or Binary Cross-Entropy Loss, is a commonly used loss function in machine learning, particularly in binary classification problems. It is designed to measure the dissimilarity between the predicted probability distribution and the true binary labels of a dataset.

Binary cross entropy compares each of the predicted probabilities to actual class output which can be either 0 or 1. It then calculates the score that penalizes the probabilities based on the distance from the expected value. That means how close or far from the actual value.

**26. How do you choose the appropriate loss function for a given problem?**

There are various factors involved in choosing a loss function for specific problem such as type of machine learning algorithm chosen, ease of calculating the derivatives and to some degree the percentage of outliers in the data set

**27. Explain the concept of regularization in the context of loss functions.**

Regularization refers to techniques that are used to calibrate machine learning models in order to minimize the adjusted loss function and prevent overfitting or underfitting.

**28. What is Huber loss and how does it handle outliers?**

Huber regression is an example of a robust regression algorithm that assigns less weight to observations identified as outliers. To do so, it uses the Huber loss in the optimization routine. Here’s a better look at what is actually happening in this model.

The Huber loss identifies outliers by considering the residuals, denoted by z. If the observation is considered to be regular (because the absolute value of the residual is smaller than some threshold \epsilon), then apply the squared loss function. Otherwise, the observation is considered to be an outlier and you apply the absolute loss. Having said that, Huber loss is basically a combination of the squared and absolute loss functions.

**29. What is quantile loss and when is it used?**

Machine learning models work by minimizing (or maximizing) an objective function. An objective function translates the problem we are trying to solve into a mathematical formula to be minimized by the model. As the name suggests, the quantile regression loss function is applied to predict quantiles. A quantile is the value below which a fraction of observations in a group falls. For example, a prediction for quantile 0.9 should over-predict 90% of the times.

Given a prediction yi^p and outcome yi, the mean regression loss for a quantile q is

**31. What is an optimizer and what is its purpose in machine learning?**

Optimization algorithms are a key part of the training process for deep learning models. They are responsible for adjusting the model parameters to minimize the loss function, which measures how well the model can make predictions on a given dataset. Different optimization algorithms are available, and choosing which can significantly impact the model's performance.

**32. What is Gradient Descent (GD) and how does it work?**

Gradient Descent Algorithm iteratively calculates the next point using gradient at the current position, scales it (by a learning rate) and subtracts obtained value from the current position (makes a step). It subtracts the value because we want to minimise the function (to maximise it would be adding). This process can be written as:

There’s an important parameter η which scales the gradient and thus controls the step size. In machine learning, it is called learning rate and have a strong influence on performance.

The smaller learning rate the longer GD converges, or may reach maximum iteration before reaching the optimum point

If learning rate is too big the algorithm may not converge to the optimal point (jump around) or even to diverge completely.

In summary, Gradient Descent method’s steps are:

choose a starting point (initialisation)

calculate gradient at this point

make a scaled step in the opposite direction to the gradient (objective: minimise)

repeat points 2 and 3 until one of the criteria is met:

maximum number of iterations reached

step size is smaller than the tolerance (due to scaling or a small gradient).

Below, there’s an exemplary implementation of the Gradient Descent algorithm (with steps tracking):

**34. What is the learning rate in GD and how do you choose an appropriate value?**

The learning rate hyperparameter controls the rate or speed at which the model learns. Specifically, it controls the amount of apportioned error that the weights of the model are updated with each time they are updated

**35. How does GD handle local optima in optimization problems?**

Gradient descent is an optimization algorithm used to minimize a function iteratively. When applied to complex optimization problems, such as those with multiple local optima, gradient descent can sometimes get trapped in a suboptimal solution. However, there are a few ways to handle local optima:

Initialization: The starting point of gradient descent plays a crucial role. Different initializations can lead to different solutions. By using multiple random initializations or more intelligent initialization techniques, such as Xavier initialization or K-means++ initialization, one can increase the chances of finding a good solution that avoids local optima.

Learning rate adjustment: The learning rate determines the step size taken by gradient descent in each iteration. A high learning rate may cause overshooting and prevent convergence, while a low learning rate can lead to slow convergence. Adaptive learning rate algorithms, such as AdaGrad, RMSprop, or Adam, adjust the learning rate dynamically during training to speed up convergence and mitigate the risk of getting stuck in local optima.

Momentum: Adding momentum to the gradient descent algorithm can help overcome local optima. Momentum introduces a "velocity" term that accumulates the previous gradients and guides the current update direction. By incorporating momentum, the algorithm can continue making progress even when facing shallow or narrow local optima.

Randomness: Injecting randomness into the optimization process can help escape local optima. One approach is to introduce random noise during parameter updates, which can prevent the algorithm from getting stuck in a particular region. Another method is to use stochastic gradient descent (SGD), which randomly samples a subset (mini-batch) of training data in each iteration. This randomness adds variability to the optimization process, allowing the algorithm to explore different regions of the parameter space.

Hybrid approaches: Sometimes, a combination of different optimization algorithms can be used to handle local optima. For example, one can perform a global optimization method like genetic algorithms or simulated annealing to explore a wide range of solutions initially and then refine the best solution found using gradient descent.

It's important to note that while these techniques can help mitigate the local optima problem, they do not guarantee finding the global optimum in all cases. The choice of approach depends on the specific problem and its characteristics.

**36. What is Stochastic Gradient Descent (SGD) and how does it differ from GD?**

Computation: GD computes the gradient using all training examples, while SGD computes an estimate of the gradient using a single example or a mini-batch.

Speed: SGD is generally faster than GD because it processes only a subset of examples in each iteration, while GD processes the entire dataset.

Stochasticity: SGD introduces randomness into the optimization process due to the random sampling of examples. This randomness can help the algorithm escape local optima and find different regions of the parameter space.

Noisy updates: Since SGD uses only a subset of examples to estimate the gradient, the updates can be noisy and exhibit more fluctuation compared to GD. However, this noise can sometimes help in regularizing the model and avoiding overfitting

**37. Explain the concept of batch size in GD and its impact on training**

In Gradient Descent (GD) and its variants, the batch size refers to the number of training examples used in each iteration to compute the gradient and update the model parameters. The batch size plays a significant role in training and can have an impact on the convergence, computational efficiency, and generalization of the model.

Here are three common choices for the batch size and their effects:

Batch Gradient Descent (Batch GD): In Batch GD, the batch size is set to the total number of training examples available in the dataset. Therefore, the algorithm computes the gradient and updates the parameters using all the training examples in each iteration. This approach provides an accurate estimate of the true gradient, but it can be computationally expensive, especially for large datasets. It requires storing and processing the entire dataset, which may not fit into memory for very large datasets. However, Batch GD tends to converge to a more stable solution since it makes more precise updates based on the entire dataset.

Stochastic Gradient Descent (SGD): In SGD, the batch size is set to 1, meaning that only one randomly selected training example is used to compute the gradient and update the parameters in each iteration. SGD introduces more randomness into the optimization process due to the single example updates. It can be computationally efficient, especially for large datasets, as it only requires processing one example at a time. However, the gradient estimates can be noisy due to the high variance resulting from a single example. This noise can cause the loss function to fluctuate during training, but it can also help SGD escape local optima and explore different areas of the parameter space.

Mini-Batch Gradient Descent: Mini-batch GD uses a batch size greater than 1 but smaller than the total dataset size. It randomly samples a small subset (mini-batch) of training examples and computes the gradient based on this subset. Mini-batch GD strikes a balance between the accuracy of Batch GD and the efficiency of SGD. It provides a more stable gradient estimate compared to SGD due to the larger batch size, while still being computationally efficient by processing a subset of examples. The choice of mini-batch size depends on factors such as the dataset size, available computational resources, and the specific problem. Common mini-batch sizes range from tens to hundreds or even thousands of examples.

**38. What is the role of momentum in optimization algorithms?**

Momentum is an extension to the gradient descent optimization algorithm, often referred to as gradient descent with momentum.

It is designed to accelerate the optimization process, e.g. decrease the number of function evaluations required to reach the optima, or to improve the capability of the optimization algorithm, e.g. result in a better final result.

A problem with the gradient descent algorithm is that the progression of the search can bounce around the search space based on the gradient. For example, the search may progress downhill towards the minima, but during this progression, it may move in another direction, even uphill, depending on the gradient of specific points (sets of parameters) encountered during the search.

This can slow down the progress of the search, especially for those optimization problems where the broader trend or shape of the search space is more useful than specific gradients along the way.

One approach to this problem is to add history to the parameter update equation based on the gradient encountered in the previous updates.

This change is based on the metaphor of momentum from physics where acceleration in a direction can be accumulated from past updates

**39. What is the difference between batch GD, mini-batch GD, and SGD?**

Batch Gradient Descent (Batch GD), Mini-Batch Gradient Descent, and Stochastic Gradient Descent (SGD) are variations of the Gradient Descent (GD) optimization algorithm. They differ in terms of the number of training examples used in each iteration to compute the gradient and update the model parameters. Here are the main differences between them:

Batch Gradient Descent (Batch GD):

Batch size: The batch size in Batch GD is equal to the total number of training examples available in the dataset.

Gradient computation: Batch GD computes the gradient of the cost function with respect to all training examples in each iteration.

Parameter update: It updates the model parameters based on the sum of the gradients computed from the entire dataset.

Convergence: Batch GD tends to converge to the true global minimum of the cost function but can be computationally expensive, especially for large datasets, as it requires processing the entire dataset in each iteration.

Mini-Batch Gradient Descent:

Batch size: Mini-Batch GD uses a batch size that is greater than 1 but smaller than the total dataset size.

Gradient computation: It randomly samples a small subset (mini-batch) of training examples to compute the gradient in each iteration.

Parameter update: The model parameters are updated based on the average gradient computed from the mini-batch.

Convergence: Mini-Batch GD provides a balance between accuracy and efficiency. It converges faster than Batch GD as it provides more frequent parameter updates, while still being computationally efficient by processing a subset of examples.

Stochastic Gradient Descent (SGD):

Batch size: SGD uses a batch size of 1, which means it randomly selects a single training example in each iteration.

Gradient computation: It computes the gradient based on the randomly selected training example.

Parameter update: The model parameters are updated based on the gradient computed from the single example.

Convergence: SGD is computationally efficient, especially for large datasets, as it only processes one example at a time. However, it can have high variance in the gradient estimates due to the single example updates, which can cause the loss function to fluctuate during training. Despite the fluctuations, SGD can help escape local optima and explore different areas of the parameter space.

In summary, Batch GD computes the gradient using the entire dataset, Mini-Batch GD computes the gradient using a subset (mini-batch), and SGD computes the gradient using a single example. Batch GD provides accurate gradient estimates but can be computationally expensive. Mini-Batch GD balances between accuracy and efficiency, and SGD is computationally efficient but can have high variance in the gradient estimates. The choice of which method to use depends on the dataset size, available computationalresources, and the specific problem at hand.

**40. How does the learning rate affect the convergence of GD?**

The learning rate is a crucial hyperparameter in Gradient Descent (GD) optimization algorithms, and it significantly affects the convergence behavior of the algorithm. The learning rate determines the step size taken in each iteration to update the model parameters. Here's how the learning rate impacts convergence:

Convergence speed:

Large learning rate: A large learning rate can result in faster convergence initially since the updates to the parameters are more significant. However, if the learning rate is too large, it can cause the algorithm to overshoot the minimum and fail to converge. The updates may become unstable, oscillating around the optimal solution or even diverging.

Small learning rate: A small learning rate leads to more conservative updates, taking smaller steps towards the minimum. While this can lead to more stable convergence, it may slow down the convergence process, requiring more iterations to reach the optimal solution.

Convergence to the optimal solution:

Proper learning rate: An appropriately chosen learning rate allows GD to converge to the optimal solution. It ensures that the updates are neither too large to miss the optimal solution nor too small to slow down convergence excessively.

Incorrect learning rate: If the learning rate is too high, the algorithm may overshoot the optimal solution and fail to converge. On the other hand, if the learning rate is too small, GD may get trapped in a suboptimal solution or take an impractically long time to converge.

Stability and oscillations:

Learning rate and oscillations: A high learning rate can introduce oscillations in the optimization process. The algorithm may constantly overshoot and undershoot the optimal solution, causing fluctuations in the loss function. This can make it challenging for GD to converge or converge slowly. A lower learning rate helps to stabilize the updates and reduce oscillations.

Learning rate and plateaus: In regions of the parameter space with a small gradient (plateaus), a high learning rate can cause GD to bounce around without making significant progress. This phenomenon is known as the "zig-zagging" effect. A smaller learning rate can help GD make smoother and more precise updates, allowing it to overcome plateaus more effectively.

Robustness to noisy gradients:

Learning rate and noisy gradients: In the presence of noisy or fluctuating gradients, a smaller learning rate can help GD by reducing the impact of noisy updates and providing a smoother optimization path. It helps to prevent the algorithm from being influenced too much by individual noisy gradients and promotes more stable convergence.

Selecting an appropriate learning rate often involves experimentation and finding the right balance between convergence speed and stability. Techniques such as learning rate schedules, where the learning rate is annealed or decayed over time, can be employed to achieve faster convergence initially while allowing for more fine-grained adjustments later in the optimization process

**41. What is regularization and why is it used in machine learning?**

Regularization refers to techniques that are used to calibrate machine learning models in order 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.

**42. What is the difference between L1 and L2 regularization?**

L1 regularization, also known as L1 norm or Lasso (in regression problems), combats overfitting by shrinking the parameters towards 0. This makes some features obsolete.

L2 regularization, or the L2 norm, or Ridge (in regression problems), combats overfitting by forcing weights to be small, but not making them exactly 0.

So, if we’re predicting house prices again, this means the less significant features for predicting the house price would still have some influence over the final prediction, but it would only be a small influence.

**43. Explain the concept of ridge regression and its role in regularization.**

Ridge regression is a model tuning method that is used to analyse any data that suffers from multicollinearity. This method performs L2 regularization. When the issue of multicollinearity occurs, least-squares are unbiased, and variances are large, this results in predicted values being far away from the actual values.

Lambda is the penalty term. λ given here is denoted by an alpha parameter in the ridge function. So, by changing the values of alpha, we are controlling the penalty term. The higher the values of alpha, the bigger is the penalty and therefore the magnitude of coefficients is reduced.

It shrinks the parameters. Therefore, it is used to prevent multicollinearity

It reduces the model complexity by coefficient shrinkage

44.

**45. How does regularization help prevent overfitting in machine learning models**?

Regularization adds the penalty as model complexity increases. The regularization parameter

(lambda) penalizes all the parameters except intercept so that the model generalizes the data and

won’t overfit.

Ridge regression adds “squared magnitude of the coefficient" as penalty term to the loss

function. Here the box part in the above image represents the L2 regularization element/term

**46. What is early stopping and how does it relate to regularization?**

The idea is very simple. The model tries to chase the loss function crazily on the training data, by tuning the parameters. Now, we keep another set of data as the validation set and as we go on training, we keep a record of the loss function on the validation data, and when we see that there is no improvement on the validation set, we stop, rather than going all the epochs. This strategy of stopping early based on the validation set performance is called Early Stopping.

**47. Explain the concept of dropout regularization in neural networks.**

Dropout is a technique where randomly selected neurons are ignored during training. They are “dropped-out” randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

**50. What is the trade-off between bias and variance in regularized** models?https://towardsdatascience.com/regularization-the-path-to-bias-variance-trade-off-b7a7088b4577

**51.** **What is Support Vector Machines (SVM) and how does it work?**

SVM or Large margin classifier is a supervised learning algorithm that uses a powerful technique called SVM for classification.

:In Linear SVM, the data points are expected to be separated by some apparent

gap. Therefore, the SVM algorithm predicts a straight hyperplane dividing the two classes. The

hyperplane is also called as maximum margin hyperplane

It is possible that our data points are not linearly separable in a pdimensional space, but can be linearly separable in a higher dimension. Kernel tricks make it

possible to draw nonlinear hyperplanes. Some standard kernels are a) Polynomial Kernel b) RBF

kernel(mostly used).

**52. How does the kernel trick work in SVM?**

The kernel trick provides a solution to this problem. The “trick” is that kernel methods represent the data only through a set of pairwise similarity comparisons between the original data observations x (with the original coordinates in the lower dimensional space), instead of explicitly applying the transformations ϕ(x) and representing the data by these transformed coordinates in the higher dimensional feature space.

In kernel methods, the data set X is represented by an n x n kernel matrix of pairwise similarity comparisons where the entries (i, j) are defined by the kernel function: k(xi, xj). This kernel function has a special mathematical property

**53. What are support vectors in SVM and why are they important?**

Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier.

**54. Explain the concept of the margin in SVM and its impact on model performance.**

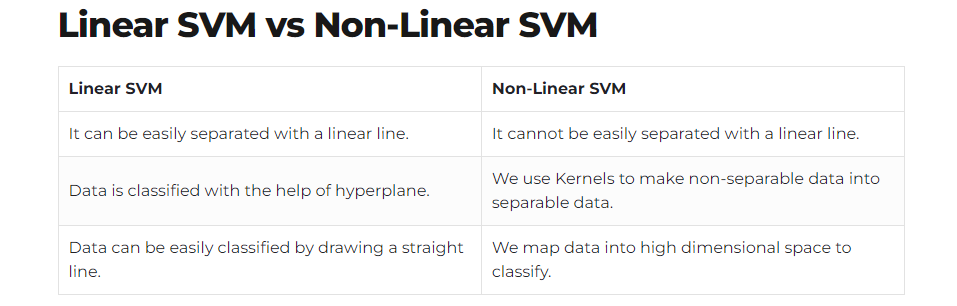
Margin: it is the distance between the hyperplane and the observations closest to the hyperplane (support vectors). In SVM large margin is considered a good margin. There are two types of margins hard margin and soft margin.

**55. How do you handle unbalanced datasets in SVM?**

The data may be transformed using a kernel to allow linear hyperplanes to be defined to separate the classes in a transformed feature space that corresponds to a nonlinear class boundary in the original feature space. Common kernel transforms include a linear, polynomial, and radial basis function transform. This transformation of data is referred to as the “kernel trick.”

Typically, the classes are not separable, even with data transforms. As such, the margin is softened to allow some points to appear on the wrong side of the decision boundary. This softening of the margin is controlled by a regularization hyperparameter referred to as the soft-margin parameter, lambda, or capital-C (“C“).

**56. What is the difference between linear SVM and non-linear SVM?**

****

**57. What is the role of C-parameter in SVM and how does it affect the decision boundary?**

The parameter C in SVM helps control the trade-off between the training error and the margin since it can determine the penalty for misclassified data points during the training process.

To be specific, a smaller value of C allows for a larger margin, potentially leading to more misclassifications on the training data. On the other hand, a larger value of C puts more emphasis on minimizing the training error, potentially leading to a narrower margin.

When the value of C is small, the SVM algorithm focuses more on achieving a larger margin. In other words, a smaller C value allows for more misclassifications in the training data, which can result in a wider margin between the classes.

Note that this can be useful in scenarios where the data points are well-separated, and there is a low presence of noise or outliers. However, it is important to be cautious as setting \bold{C} too small can lead to underfitting, where the model fails to capture the underlying patterns in the data.

Conversely, a larger value of C in SVM emphasizes minimizing the training error, potentially resulting in a narrower margin. When C is set to a large value, the SVM algorithm seeks to fit the training data as accurately as possible, even if it means sacrificing a wider margin.

Remember that this can be beneficial when the data points are not well-separated or when there is a significant presence of noise or outliers. However, setting \bold{C} too large can increase the risk of overfitting, where the model becomes too specific to the training data and performs poorly on new, unseen data.

**58. Explain the concept of slack variables in SVM.**

It allows there to be a trade off between learning "simple" functions and fitting the data exactly. Without slack variables the SVM would be forced into always fitting the data exactly and would often overfit as a result, unless fitting exactly is impossible, then there is no solution without slack variables and software that didn't include them would have to output an error in this case

**61. What is a decision tree and how does it work?**

A decision tree is a supervised machine learning algorithm used for both classification and regression tasks. It is a predictive model that makes decisions or predictions by following a tree-like structure, where each internal node represents a feature or attribute, each branch represents a decision based on that attribute, and each leaf node represents an outcome or prediction.

Here's how a decision tree works:

Tree construction: The decision tree algorithm starts with the entire dataset at the root node. It analyzes different attributes/features to find the best split that separates the data into the most homogeneous subsets based on the target variable (for classification) or minimizing the prediction error (for regression). The attribute chosen for the split is typically selected based on criteria such as information gain, Gini impurity, or mean squared error.

Recursive splitting: The tree-building process continues recursively by dividing the data into subsets based on the selected attribute and its possible values. Each subset is represented by a child node of the current node, and the process repeats until a stopping criterion is met. The stopping criterion can be reaching a maximum depth, reaching a minimum number of samples in a leaf node, or when further splitting does not significantly improve the predictive accuracy.

Leaf node assignment: Once the splitting process is complete, the tree consists of internal nodes representing attribute tests and leaf nodes representing the predicted outcomes. In classification tasks, each leaf node corresponds to a specific class label, while in regression tasks, the leaf nodes contain the predicted continuous values.

Tree pruning (optional): After constructing the decision tree, a pruning process can be applied to avoid overfitting. Pruning involves removing or collapsing branches that contribute little to the overall predictive power of the tree. This helps prevent the model from memorizing noise in the training data and promotes better generalization to unseen data.

Prediction: To make a prediction or classify a new instance, the algorithm traverses down the decision tree, following the attribute tests at each internal node based on the instance's attribute values. Eventually, it reaches a leaf node, which provides the prediction or class label for the given instance.

62.

For each split, individually calculate the entropy of each child node

Calculate the entropy of each split as the weighted average entropy of child nodes

Select the split with the lowest entropy or highest information gain

Until you achieve homogeneous nodes, repeat steps 1-3

**63. What are impurity measures (e.g., Gini index, entropy) and how are they used in decision trees?**

Gini Index

The other way of splitting a decision tree is via the Gini Index. The Entropy and Information Gain method focuses on purity and impurity in a node. The Gini Index or Impurity measures the probability for a random instance being misclassified when chosen randomly. The lower the Gini Index, the better the lower the likelihood of misclassification.

**64. Explain the concept of information gain in decision trees.**

Information gain is a metric used in decision trees to evaluate the quality of a potential split based on the information theory concept of entropy. It measures the reduction in uncertainty or randomness in the target variable (class labels) after splitting the data on a particular attribute.

**65. How do you handle missing values in decision trees?**

There are several methods used by various decision trees. Simply ignoring the missing values (like ID3 and other old algorithms does) or treating the missing values as another category (in case of a nominal feature) are not real handling missing values. However those approaches were used in the early stages of decision tree development.

The real handling approaches to missing data does not use data point with missing values in the evaluation of a split. However, when child nodes are created and trained, those instances are distributed somehow.

I know about the following approaches to distribute the missing value instances to child nodes:

all goes to the node which already has the biggest number of instances (CART, is not the primary rule)

distribute to all children, but with diminished weights, proportional with the number of instances from each child node (C45 and others)

distribute randomly to only one single child node, eventually according with a categorical distribution (I have seen that in various implementations of C45 and CART for a faster running time)

build, sort and use surrogates to distribute instances to a child node, where surrogates are input features which resembles best how the test feature send data instances to left or right child node (CART, if that fails, the majority rule is used)

**66. What is pruning in decision trees and why is it important?**

An alternative method to prevent overfitting is to try and stop the tree-building process early, before it produces leaves with very small samples. This heuristic is known as early stopping but is also sometimes known as pre-pruning decision trees.

At each stage of splitting the tree, we check the cross-validation error. If the error does not decrease significantly enough then we stop. Early stopping may underfit by stopping too early. The current split may be of little benefit, but having made it, subsequent splits more significantly reduce the error.

Early stopping and pruning can be used together, separately, or not at all. Post pruning decision trees is more mathematically rigorous, finding a tree at least as good as early stopping. Early stopping is a quick fix heuristic.

**.68. How do you interpret the decision boundaries in a decision tree?**

**How was this decision boundary decided ?**

A decision boundary is decided by testing all the possible decision boundaries splitting the dataset and choosing the one that minimizes the Gini impurity of the two splits.

What is Gini impurity ?

Gini impurity is a metric that measures the probability from a randomly chosen element (here an iris) to be incorrectly classified, i.e. the probability of choosing an element times the probability of being misclassified. If we sum over all J possible classes we have the Gini impurity :

The last expression is the one we are going to use to perform the Gini test.

Let’s compute the Gini impurity for the first node

At the root node all the data points are mixed. Using the result above Gini impurity is :

This gives us :

We can verify this number by checking the Gini information on the root node : ‘gini = 0.664’. For the first node we have a Gini impurity of 0.664.

Let’s get back to the first decision boundary

The question to be asked to determine a decision boundary is : how to split the iris species so that we create more homogeneous groups ?

Intuitively what we can observe on the graph above is that we can create a homogeneous group containing only setosa species just by splitting the dataset along the petal width axis.

But the algorithm has no intuition. So how does it find the best split ?

It will try all the possible boundaries along all the features, i.e. all the axes petal width and sepal width.

For each split the algorithm will compute the Gini impurity of the two groups created.

Finally it will choose the decision boundary that gives the lowest Gini impurity for the two groups (either summing the Gini impurity for each group or doing a mean).

Let’s get back to the first node and the first split

In the case of the root node, the algorithm has found that among all the possible splits the split with petal width = 0.8 cmgives the lowest Gini impurity.

The Gini impurity for the left leaf is :

We verify this result with the tree graph. This result is not surprising because in the left leaf which matches the left part of the graph we only have setosa iris, so the group is very homogeneous and Gini impurity is a measure of homogeneity.

**70. What are ensemble techniques and how are they related to decision trees?**

Ensemble techniques, such as bagging and boosting, offer a powerful approach to enhance the performance of decision trees in machine learning tasks. By combining the predictions of multiple decision trees, these techniques increase accuracy, improve robustness, and provide valuable insights into feature importance