Analityka Danych - opracowanie pytań

Autor: Dawid Włosek
Styczeń 2024

Data Exploration

1. Explain what are the numerical and categorical variables.

Numerical variables represent measurable quantities with a numerical value. They can be further classified into discrete and continuous variables.

- Can take any numerical value within a range.
- Operations like addition, subtraction, multiplication, and division make sense.
- Data can be ordered or ranked based on the numerical values.

Categorical variables represent categories or labels and are used to classify data into groups. They can be further classified into nominal and ordinal variables.

- Represents distinct categories or groups.
- No inherent order among categories in nominal variables.
- Ordinal variables have a meaningful order or ranking among categories, but the intervals between categories may not be uniform.
- Operations like addition and subtraction are not meaningful for categorical variables.

2. Explain what is the difference between experimental studies and observations. What it is a confounding variable. When we can draw conclusion about correlations and when about causation?

Experimental Studies:

- *Definition:* Experimental studies involve the manipulation of an independent variable to observe its effect on a dependent variable while controlling other variables.
- *Control:* Researchers can control and manipulate variables to establish cause-and-effect relationships.
- *Randomization:* Random assignment helps minimize bias and ensures that groups are comparable at the beginning of the study.
- *Example:* A clinical trial testing the effectiveness of a new drug, where participants are randomly assigned to receive the drug or a placebo.

Observational Studies:

- *Definition:* Observational studies involve the observation and measurement of variables without intervention or manipulation by the researcher.
- *Control:* Researchers do not control the variables, and causation is more challenging to establish due to the potential for confounding variables.
- *Example:* Studying the relationship between smoking and lung cancer by observing individuals over time without intervening.

Confounding variable is an additional variable that is not the main focus of the study but can affect the observed relationship between the independent and dependent variables. Confounding variables can lead to spurious associations or obscure the true relationship between variables.

Correlation refers to statistical association between two variables. It does not imply causation. Correlation suggests a relationship but does not provide evidence of a cause-and-effect link.
Causation implies a cause-and-effect relationship, where changes in one variable directly cause changes in another. Causation can be inferred when experimental designs are used, and these criteria are met.

3. What is the difference between census and the sampling? What are the sampling methods? What are the possible sources of the sampling bias.

Census involves collecting data from every individual or unit in the entire population.
Sampling involves selecting a subset of individuals or units from the population and using their data to make inferences about the entire population.

Sampling methods:

- Random sampling: Every individual in the population has an equal chance of being selected.
- Stratified sampling: The population is divided into homogenous subgroups (strata), and random samples are taken from each stratum.
- Cluster sampling: The population is divided into clusters, and random sample of clusters is selected. All individuals within the chosen clusters are included in the sample.

Possible source of the sampling bias:

- Convenience sample bias: Individuals who are easily accessible are more likely to be included in the sample.
- Non-response bias: If only a (non-random) fraction of the randomly sampled people respond to a survey, than the sample is no longer representative of the population.
- Volunteer bias: Occurs when the sample consists of people who volunteer to respond because they have strong opinions on the issue.

4. How do we define point estimate of sample statistics: measures of center and spread. What does it mean "robust statistics". Which definitions have this features.

Measures of Center:

- Mean (Arithmetic Average): The sum of all observations divided by the number of observations.
- Median: The middle value when observations are ordered. It is less sensitive to extreme values than the mean.

Measures of Spread:

- Standard Deviation: A measure of the average deviation of individual observations from the mean.
- Range: The difference between the maximum and minimum values in a dataset.
- Interquartile Range (IQR): The range of the middle 50% of the data, defined by the difference between the first quartile (Q1) and the third quartile (Q3).
- Variance: Quantifies how much individual data points differ from the mean of the dataset.

Robust statistics - statistical measures that are not heavily influenced by extreme values or outliers in a dataset. They perform well even in the presence of skewed or non-normally distributed data and provide more stable estimates when dealing with atypical observations.

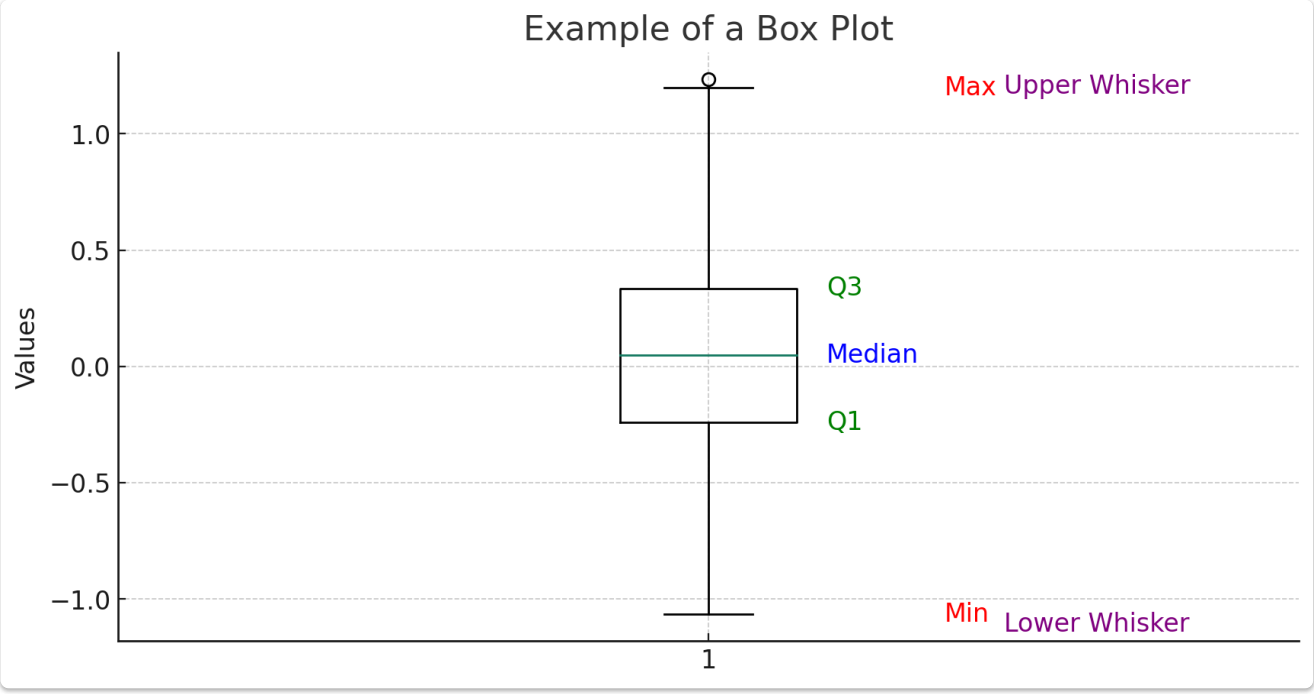
Examples of Robust Statistics:

1. *Median*: As a measure of central tendency, the median is robust because it is not affected by extreme values. It represents the middle value, making it less sensitive to outliers.

2. *Interquartile Range*: The IQR, which measures the spread of the central 50% of the data, is less affected by extreme values than the standard deviation.
3. *MAD (Median Absolute Deviation)*: The Mad is robust alternative to the standard deviation. It measures the dispersion of data point relative to the median.
4. *Trimmed Mean*: Calculated by removing a certain percentage of extreme values (e.g., 5% from both ends) before computing the mean. This helps mitigate the impact of outliers.

5. What is the box plot. Could you draw example and explain meaning of its content. How we deduce that distributions has a tail, outlayers, etc.

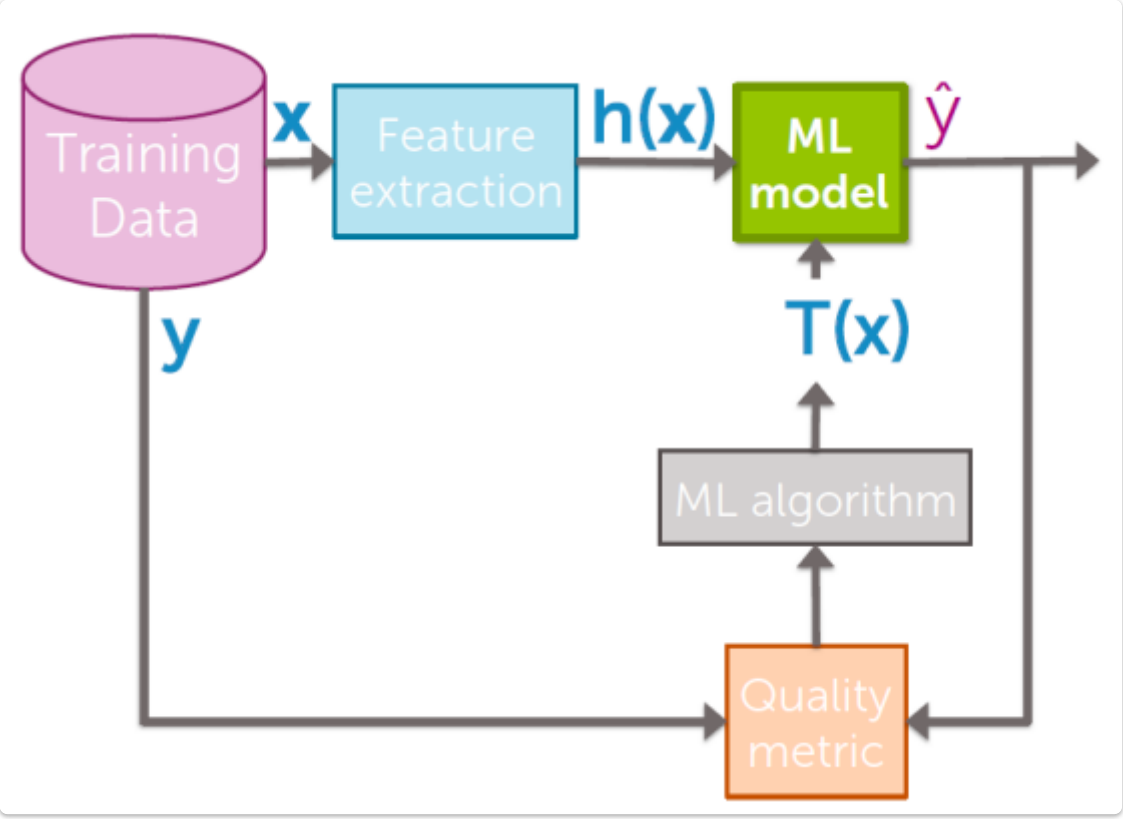
Box plot - graphical representation of the distribution of a dataset. It provides a summary of key statistical measures and helps identify the presence of outliers, the central tendency and the spread of the data.



- **Median (Green Line inside the box)**: This is the middle value of the dataset, dividing the data into two equal halves. In the plot, this is shown by the line inside the box.
- **First Quartile (Q1)**: This is the median of the lower half of the dataset (excluding the median if there is an odd number of data points). It marks the 25th percentile and is the bottom line of the box.
- **Third Quartile (Q3)**: This is the median of the upper half of the dataset. It marks the 75th percentile and is the top line of the box.
- **Interquartile Range (IQR)**: This is the range between the first and third quartiles (the height of the box). It contains the middle 50% of the data.
- **Minimum (Min)**: The lowest value within 1.5 IQR of the first quartile. It's the end of the lower whisker.
- **Maximum (Max)**: The highest value within 1.5 IQR of the third quartile. It's the end of the upper whisker.
- **Lower Whisker**: Extends from Q1 to the minimum value in the dataset that is not considered an outlier.
- **Upper Whisker**: Extends from Q3 to the maximum value that is not considered an outlier.
- **Outliers**: Data points that lie beyond the whiskers (not shown in this plot). They are typically more than 1.5 IQRs from the quartiles.

Regression

1. Draw the flow-chart diagram for making predictions using regression as ML algorithm. Explain briefly each box on the flow-chart: ML model, ML algorithm, Quality metric, Feature extraction.



Feature extraction: The process of transforming raw data into numerical features that can be used in model training. This may involve selecting certain attributes, encoding categories, scaling and more.
ML algorithm: The specific computational process used to train the machine learning model. For regression, this could be linear regression, decision trees or neural networks among others.
ML model: The result of the ML algorithm being applied to the training data. It is a mathematical representation of the learned patterns.
Quality metric: A measure used to evaluate the performance of the machine learning model, such as Mean Squared Error for regression models.

2. Write down formula for simple linear regression model. How one can interpret coefficients. Write formula for defining cost function using RSS estimator.

The general form of the simple linear regression model is:

$$Y = \beta_0 + \beta_1 X + \epsilon$$

where:

- Y is the dependent variable (the variable we are trying to predict).
- X is the independent variable (the variable used for prediction).
- β_0 is the y-intercept (the value of Y when X is 0).
- β_1 is the slope of the line (the change in Y for a one-unit change in X).
- ϵ is the error term, representing unobserved factors that affect Y but are not included in the model.

The cost function for simple linear regression is often defined using the Residual Sum of Squares (RSS) estimator. The residuals are the differences between the observed Y values and the values predicted by the model. The RSS is the sum of the squared residuals. The formula for the RSS is:

$$RSS = \sum_{i=1}^{n}(Y_i - (\beta_0 + \beta_1 X_i))^2$$

where:

- n is the number of observations.
- Y_i is the observed value of Y for the ith observation.
- X_i is the observed value of X for the ith observation.
- β_0 and β_1 are the coefficients to be estimated.

3. Write down sequence of iterative gradient descent algorithm finding minimum of the cost function for simple linear regression. When do we stop iteration, how do we choose step-size.

1. **Initialization:** Start with initial guesses for the parameters of the linear regression model. These parameters are typically slope (m) and intercept (b) of the line $y = mx + b$. The initial values could be zeros or small random numbers.
2. **Cost Function:** Define the cost function that measures the performance of the model for given parameters. For linear regression, the cost function is usually the Mean Squared Error which is the average of the squared differences between the predicted and actual values:

   $$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - (mx_i + b))^2$$

   where N is the number of observations, y_i is the actual values, and $(mx_i + b)$ is the predicted value.
3. **Calculate Gradient:** Compute the gradient of the cost function with respect to each parameter. The gradients indicate the direction of the steepest ascent in the cost function. For MSE in linear regression, the partial derivatives are:

   $$\frac{\partial MSE}{\partial m} = \frac{-2}{N}\sum_{i=1}^{N} x_i(y_i - (mx_i + b))$$

   $$\frac{\partial MSE}{\partial b} = \frac{-2}{N}\sum_{i=1}^{N}(y_i - (mx_i + b))$$

4. **Update Parameters:** Update the parameters in the direction that reduces the cost function. This is done by subtracting a fraction of the gradient from the current values of the parameters:

   $$m = m - \alpha\frac{\partial MSE}{\partial m}$$

   $$b = b - \alpha\frac{\partial MSE}{\partial b}$$

   where α is the learning rate, a small positive value that controls the size of the step.
5. **Iterate:** Repeat steps 3 and 4 until the algorithm converges.
6. **Convergence:** When the algorithm converges, the values of m and b should be at or near the values that minimize the cost function.
   Throughout the iterations, the algorithm moves the parameters m and b towards the values that will have the lowest cost (MSE).

Possible stop conditions:

1. *Convergence Threshold:* The algorithm stops when the change in the cost function between iterations falls below a predefined threshold.
2. *Maximum Iterations:* A predetermined number of iterations is set, and the algorithm stops when this limit is reached. This prevents the algorithm from running indefinitely, especially in cases where it might not converge due to various reasons.

3. *Gradient Norm:* The algorithm can be set to stop when the norm of the gradient falls below a certain threshold. A small gradient norm suggests that the cost function is near a minimum.
4. *Stability of Solution:* Sometimes, the algorithm stops when the solution (i.e., the parameters of the linear regression model) remains relatively stable over a number of iterations, indicating that further changes are minor.
5. *Validation Performance:* In some implementations, especially when dealing with machine learning, the algorithm might stop based on the performance on a validation dataset. If the performance (e.g., validation error) stops improving or starts worsening, it might be a sign to stop the gradient descent to avoid overfitting.

The **learning rate** (step-size) α is a crucial hyperparameter that must be chosen carefully, as to large of a value can cause the algorithm to overshoot the minimum, and too small of a value can lead to a very slow convergence.

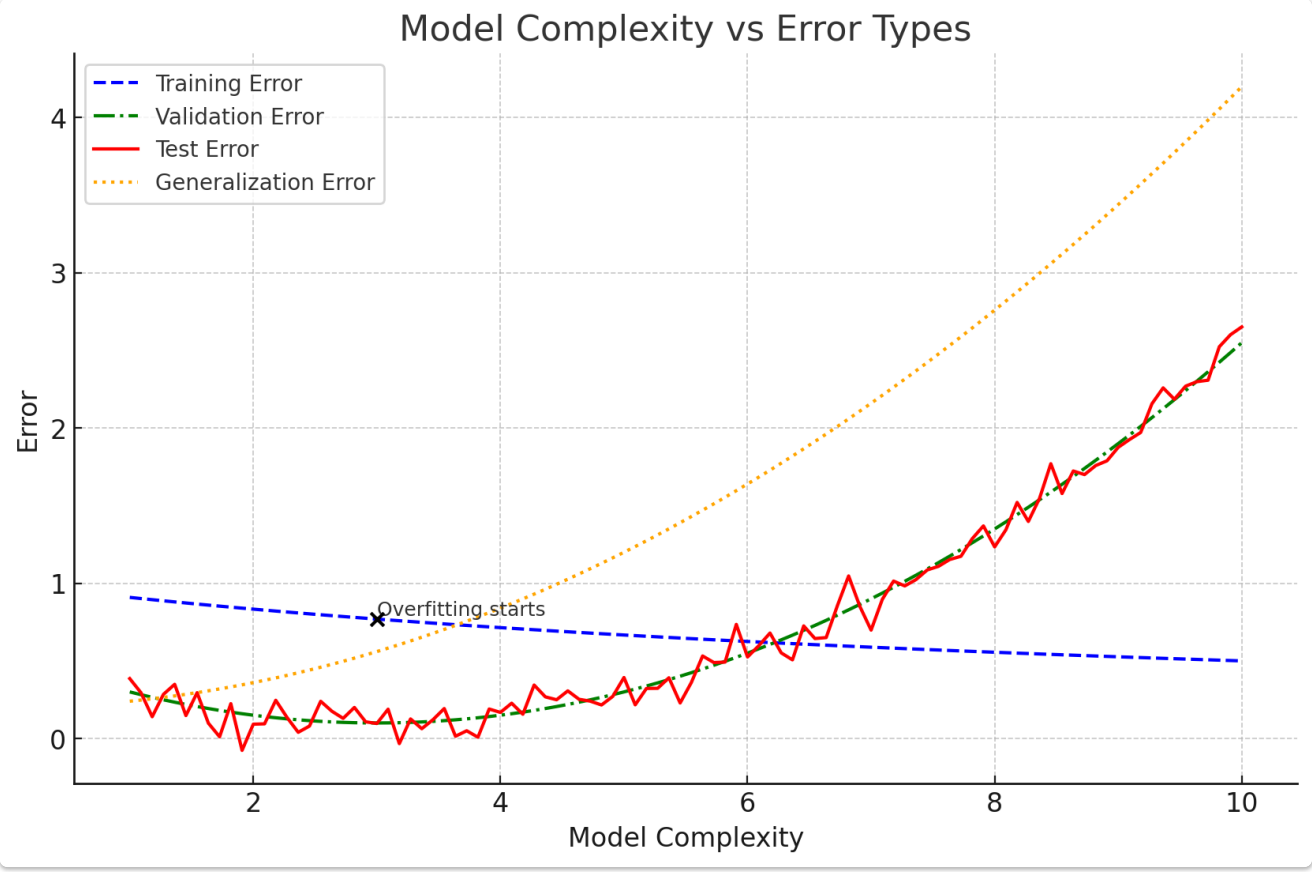
1. Start with a small value
2. Use a Learning Rate Schedule: Implement a learning rate schedule that decreases the learning rate over time. This can be done by using techniques like learning rate decay, where the learning rate is reduced by a certain factor after a certain number of epochs.
3. Try different values: Use a range of learning rates and compare the performance of the algorithm.
4. Grid Search or Random Search: Conduct a grid search or random search over a range of learning rates and possibly other hyperparameters to find the most effective combination.
5. **Use Learning Rate Optimization Algorithms**: Implement advanced optimization algorithms that can adapt the learning rate during training (e.g., AdaGrad, RMSProp, Adam). These algorithms adjust the learning rate for each parameter dynamically.
6. **Learning Rate as a Hyperparameter**: Treat the learning rate as a hyperparameter and use cross-validation to find the best value.

4. How do we access performance? Explain what is the "training error", "validation error", "generalization error", "test error". What does it mean "cross-validation"? Draw illustrative plot how they typically behave with regression model complexity.

Performance Metrics:

- Training Error: The error or loss calculated on the data used to train the model. It measures how well the model fits the training data.
- Validation Error: The error or loss calculated on a separate dataset (validation set) that was not used during training. It provides an estimate of how well the model will generalize to new, unseen data.
- Generalization Error: The difference between the training error and the validation error. It represents how well the model generalizes to new data compared to its performance on the training data.
- Test Error: The error or loss calculated on an independent dataset (test set) that was not used during training or model selection. It serves as a final assessment of the model's performance.

Cross-Validation - a technique used to assess how well a model will generalize to an independent dataset. It involves partitioning the dataset into multiple subsets (folds), training the model on some folds, and validating it on the remaining folds. This process is repeated, and performance metrics are averaged over the folds.



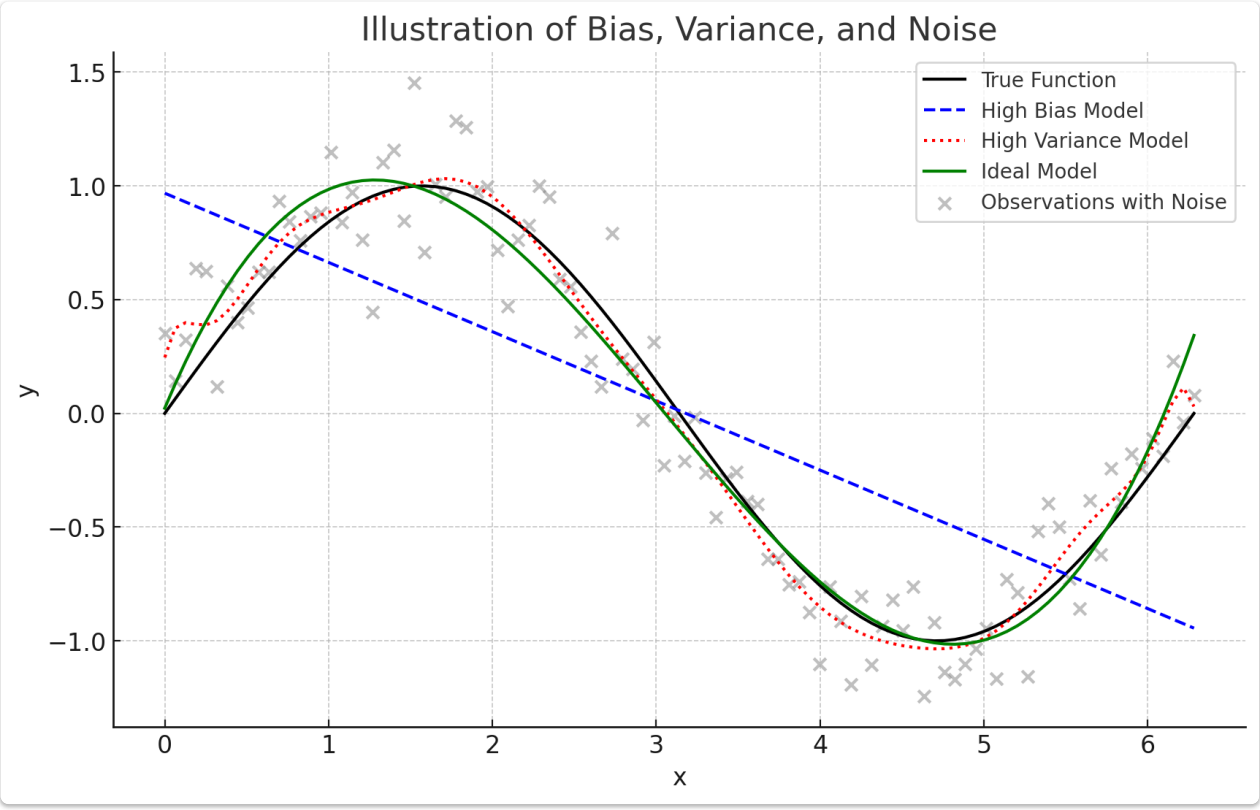
5. Explain what are the sources of errors on the prediction: noise, bias, variance. Draw simple illustration explaining it. What does it mean bias-variance trade-off.

In the context of predictive modeling, errors can be decomposed into three main sources: bias, noise and variance. These sources of error help us understand why a predictive model might be underperforming and guide us on how to improve it.

1. **Noise**
   - *Definition*: Noise refers to the inherent randomness or fluctuations in the data that cannot be modeled or predicted.
   - *Source:* It is caused by factors that are either unknown or cannot be measured and can be thought of as the background level of unpredictability in the data.
   - *Impact:* Noise sets a theoretical limit on the accuracy of predictions. Even the best model cannot predict random error that has no relation to the variables or features being studied. No matter how complex the model, noise will always contribute some level of error to the predictions.
2. **Bias**
   - *Definition:* Bias is the error introduced by approximating a real-world problem, which may be complex, by a much simpler model. In statistics, "bias" is a systematic deviation from the truth in a certain direction. A high bias error means that the model is consistently making wrong assumptions in its predictions due to oversimplification.
   - *Source:* The model being too simple to capture the relevant relationships between variables (underfitting).
   - *Impact:* These models do not pay enough attention to the training data and are unable to capture the underlying trends properly.
3. **Variance**

   - *Definition:* Variance is the error from sensitivity to small fluctuations in the training set. High variance indicates that the model learns the random noise in the training data.
   - *Source:* It is characterized by a model that models the random noise in the training data rather than the intended outputs (overfitting).
   - *Impact:* A high-variance model will perform very well on its training data but poorly on unseen data

because it's too tailored to the particularities of the training set. This model captures random noise instead of the underlying data pattern.



These sources of error are related to the concept of model complexity:

- A simple model will generally have high bias and low variance because it may be too rigid to fit the data well (think of a straight line trying to fit to a quadratic relationship).
- A complex model will have a low bias but high variance because it fits the training data too closely, including the noise.

Bias-variance trade off: Ideally, we want a model that balances bias and variance, minimizing the total error. A good model is one that captures the underlying patterns in the data (low bias) without being swayed by random fluctuations (low variance).

6. What does it mean "over-fitting"? Explain how we can mitigate it adding extra term to the cost function: "regge regression" or "lasso regression". Write formula of the respective cost functions. Show illustrative plot how the coefficients w will behave in each case.

Over-fitting happens when a model models the training data to well. It happens when a model learns both the underlying pattern and the noise in the training data to such extent that it negatively impacts the model's performance on new, unseen data, making it less generalizable or predictive.

To mitigate overfitting, one approach is to add a regularization term to the cost function. This technique penalizes the model for having too large coefficients (weights), which often corresponds to a more complex or flexible model. There are mainly two types of regularization methods used:

1. **Ridge Regression (L2 regularization):** It adds a penalty to the square of the magnitude of coefficients. The cost function for ridge regression is:

$$J(w) = \sum_{i=1}^{n}(y_i - w^T x_i)^i + \lambda \sum_{j=1}^{m} w_j^2$$

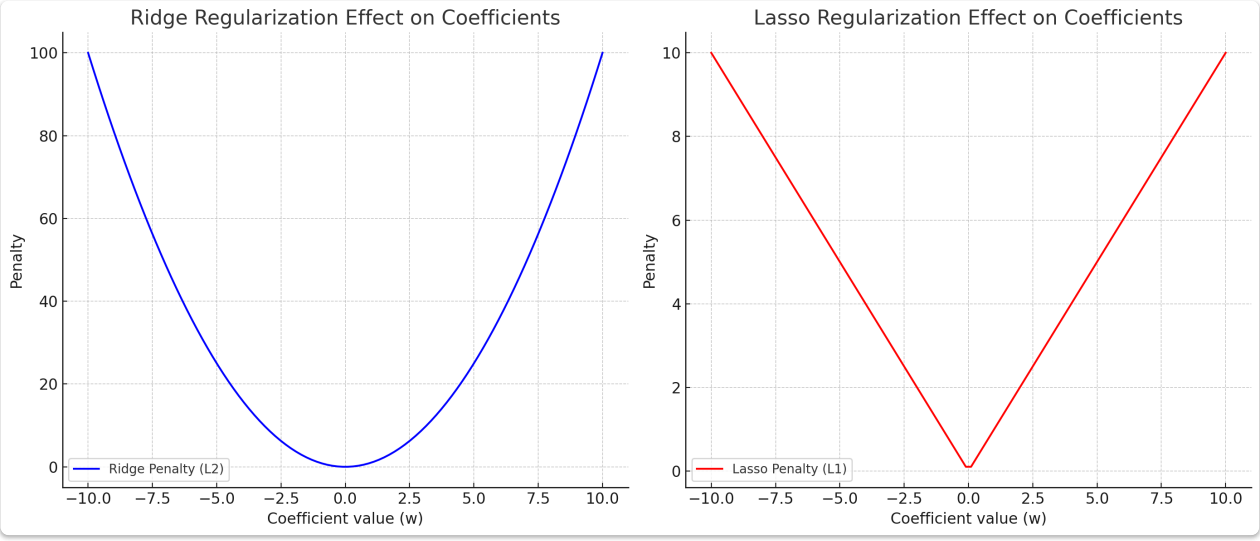
Here, the $J(w)$ is the cost function, w represents the vector of weights, x_i is the vector of features for

the i-th data point, y_i is the actual output for the i-th data point, n is the number of data points, m is the number of features, and λ is the regularization parameter.

2. **Lasso Regression (L1 regularization):** Lasso adds a penalty term that is equal to the absolute value of the magnitude of coefficients, which can lead to some coefficients being exactly zero, effectively performing feature selection. The cost function for lasso regression is:

$$J(w) = \sum_{i=1}^{n}(y_i - w^T x_i)^i + \lambda \sum_{j=1}^{m} |w_j|$$

Ridge regression is more likely to result in small, non-zero coefficients, while Lasso may result in zero coefficients, effectively reducing the number of features the model uses.



7. Explain procedure for selecting features for regression with the greedy algorithm.

Feature selection for regression using a greedy algorithm is a process of iteratively selecting a subset of relevant features for use in model construction. It involves evaluating and adding features to the model based on their statistical significance in improving the model's performance. The greedy algorithm for feature selection typically follows a forward selection, backward elimination, or a combination of both called stepwise selection.

Forward Selection:

1. *Start with no features:* Initially, the model starts with no features. This means that the initial model only includes the intercept.
2. *Evaluate Each Feature Individually:* In each step, the algorithm evaluates the contribution of each remaining feature (i.e., those not already included in the model) by adding them individually to the model and assessing their impact. The impact is usually measured using a statistical criterion, such as the reduction in RSS.
3. *Select the Best Feature:* The feature that provides the most significant improvement to the model (according to the chosen criterion) is then added to the model.
4. *Repeat the Process:* Steps 2 and 3 are repeated, each time considering only the remaining features that are not yet in the model. In each iteration, the feature that most improves the model is added.
5. *Stopping Criterion:* This process continues until a stopping criterion is reached. This criterion could be a set number of features, a threshold for improvement metric (like a minimal decrease in error), or when adding a new feature no longer significantly improves the model.

Backward stepwise: Start with full model and iteratively remove features least useful to fit.
Combining forward and backward steps: In forward algorithm, insert steps to remove features no longer as important.

8. What does it mean "non-parametric regression". Explain concept of: (1-NN) regression, (k-NN) regression, weighted regression, kernel regression.

Non-parametric regression refers to a family of models that do not assume a fixed form for the relationship between the independent variables and the dependent variable. Unlike parametric models, which have a predetermined number of parameters, non-parametric models are flexible and can fit a large number of shapes of the data.

1. **(1-NN) regression (One-Nearest Neighbour Regression):**
   - In 1-NN regression, the predicted value for a new observation is simply the values of the dependent variable for the closest observation in the training set.
   - This is determined by some distance measure, often Euclidean distance.
   - It's a very simple form of non-parametric regression but can be highly sensitive to noise in the data since it relies on only one nearest neighbour.
2. **(k-NN) regression (k-Nearest Neighbours Regression):**
   - This method extends 1-NN by considering the 'k' nearest neighbours.
   - The predicted value is typically the average (or sometimes the median) of the values of these neighbours.
   - The choice of 'k' affects the bias-variance tradeoff: a small 'k' can lead to overfitting (high variance), while a large 'k' can smooth out peculiarities in the data, potentially leading to underfitting (high bias).
3. **Weighted regression:**
   - In the context of k-NN, a weighted version gives more importance to closer neighbours than to further ones.
   - Weights can be determined in various ways, often decreasing with distance. A common scheme is to use the inverse of distance as the weight, so that closer points have a larger influence on the prediction.
4. **Kernel Regression:**
   - Kernel regression is a type of weighted average regression that uses kernel functions to weigh the observations.
   - The weight of each observation is determined by its distance from the point of interest, with closer point receiving higher weights.
   - The kernel function defines the shape of the weigths (e.g., Gaussian, Epanechnikov). It falls off smoothly and symmetrically around the prediction point, ensuring that points further away have less influence.

These techniques are used to predict a continuous outcome variable and can be particularly powerful when the true relationship between the independent and dependent variables is complex and unknown.

Classification

1. Explain model of logistic regression classifier. Write down formula for linear score and logistic link function. How it is extended in case of multi-classification problem.

The logistic regression classifier is a statistical model used for binary classification. It predicts the probability that a given input point belongs to one of the two categories based on a logistic function.
The model works as follows:

1. **Linear Score (Linear Combination):** First, logistic regression computes a linear score (also known as the logit or log-odds) by applying a linear function to the input features. This score is the weighted sum of the input features (X) plus a bias term (intercept), often denoted as z.
   The formula for the linear score is $z = w^T x + b$, where:

- w is the weight vector (coefficients),
- x is the input feature vector,
- b is the bias term (intercept).

2. **Logistic (Sigmoid) Link Function:** The linear score is then transformed using the logistic (sigmoid) function to produce a probability value between 0 and 1.
   The logistic function is defined as: $\sigma(z) = \frac{1}{1+e^{-z}}$, where:
   - $\sigma(z)$ denotes the logistic function,
   - e is the base of the natural logarithm.
     The output of the logistic function gives the probability that the input X belongs to the positive class (usually labeled as '1'), which can be written as:

$$P(Y = 1|X) = \sigma(z)$$

For **multi-class classification** problems, logistic regression can be extended using one of the following approaches:

1. **One-vs-Rest (OvR) or One-vs-All (OvA):** In this approach, a separate logistic regression classifier is trained for each class to distinguish that class from all other classes. For a problem with K classes, K different classifiers are trained. When making predictions, the class corresponding to the classifier with the highest probability is chosen.
2. **Multinomial Logistic Regression:** This is a direct extension of binary logistic regression to multi-class problems without having to train multiple binary classifiers. The model uses a *softmax* function instead of a sigmoid to handle multiple classes. The *softmax* function is defined as:

$$P(Y = k|X) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

   where:
   - $P(Y = k|X)$ is the probability that input X belongs to class k,
   - z_k is the score for class k computes as $z_k = w_k^T x + b_k$,
   - K is the total number of classes.
     In the *softmax* function, z_k is computed for each class, and then the probabilities are normalized across all classes so that they sum up to 1. The class with the highest probability is the predicted class.

2. We measure performance of the classifier based on: "classification error", "classification accuracy", "confusion matrix". Could you explain what does it mean? What is the problem of "class majority".

Classification Error:

- Classification error is the proportion of the total number of predictions that were incorrect. It measures how often the classifier makes mistakes. The formula to calculate it is:

$$Classification Error = \frac{Number - of - Incorrect - Predictions}{Total - Number - of - Predictions}$$

- A higher classification error indicates worse performance of the classifier.

Classification Accuracy:

- Classification accuracy is the proportion of the total number of predictions that were correct. It's the most intuitive performance measure and it is simply the ratio of correctly predicted observations to the total observations.

$$Classification Accuracy = \frac{Number - of - Correct - Predictions}{Total - Number - of - Predictions}$$

- High accuracy indicates a better model, but it can be misleading if the data set is unbalanced (which leads to the issue of "class majority").

Confusion Matrix:

- A confusion matrix is a table used to describe the performance of a classification model on a set of test data for which the true values are known. It allows you to visualize the classifier's performance.
- The matrix compares the actual target values with those predicted by the machine learning model. It gives insights not just into errors being made by a classifier but more importantly the types of errors that are being made.
- In a binary classification, the table has 2 rows and 2 columns, with one axis of the matrix representing the actual classes and the other the predicted classes. It includes terms like *True Positives (TP)*, *True Negatives (TN)*, *False Positives (FP)*, and *False Negatives (FN)*

Actual \ Predicted	Positive Prediction	Negative Prediction
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

In each cell of a confusion matrix, we put the count of instances that fall into the corresponding category.

Class Majority: The problem of class majority, or class imbalance, occurs when the number of observations in one class significantly outweighs the number of observations in other classes. This can cause the following issues:

- *Misleading Accuracy:* A classifier might simply predict the majority class for all inputs, achieving high accuracy but poor predictive performance.
- *Poor Generalization:* The classifier may not perform well in predicting the minority class, which is often the class of greater interest.
- *Biased Models*: Learning is biased towards the majority class, ignoring the characteristics of the minority class.

Solutions to class majority problems include resampling the dataset to balance the classes, using performance metrics that give a better picture of model performance across classes (like F1-score, precision, recall), or using algorithmic techniques that place more emphasis on the minority class.

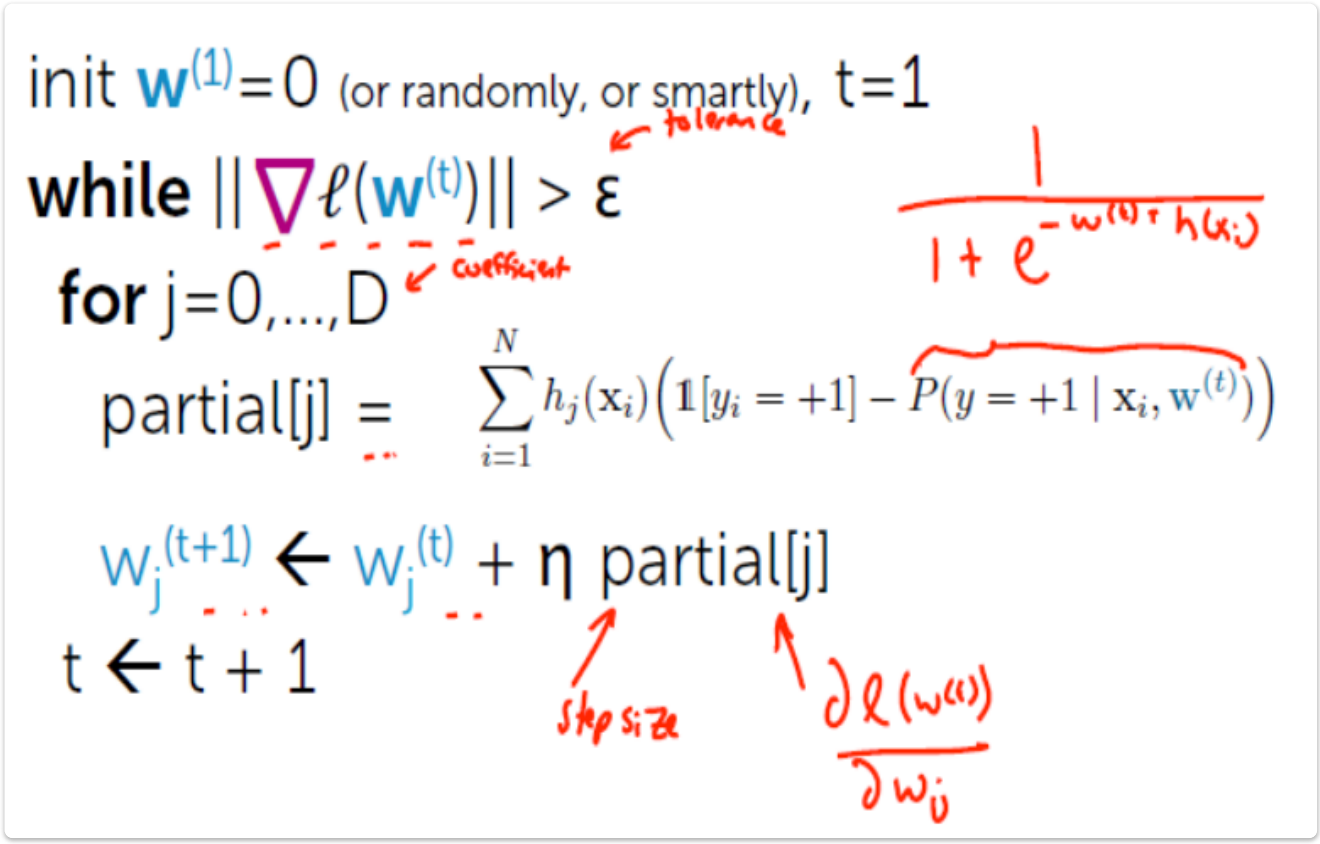
3. Write formula for quality metric in case of logistic classifier: likelihood function. The best classifier is found using MLE (maximum likelihood estimation) method and gradient ascent. Could you write down and explain final formula of that algorithm. How do we choose step size.

Likelihood $l(w)$ **:** Measures quality of fit for model with coefficients w:

$$l(w) = \prod_{i=1}^{N} P(y_i|x_i, w)$$

Using gradient ascent for finding the best classifier is about finding the best model coefficients w with

gradient ascent:



where η is the step size.

Choosing step size:

- Unfortunately, picking step size requires a lot of trial and error
- Try several values, exponentially spaced
  - *Goal:* plot learning curves to:
    - find one η that is too small (smooth but moving too slowly)
    - find one η that is too large (oscillation or divergence)
- Try values in between to find "best" η
  - values need to be exponentially spaced, pick one that leads to the best training data likelihood
- Advanced tip: can also try step size that decreases with iterations, e.g., $\eta_t = \frac{\eta_0}{t}$

4. Classification with decision trees. How one defines classification of the final leafs. How one measure quality of the predictions: error and accuracy. Explain simple greedy algorithm to find the best decision tree. How one is measuring performance.

Classification of the Final Leaves:

- In a decision tree for classification, the final leaves (also called terminal nodes) represent the categories or classes into which the input data points are classified.
- The class assigned to a leaf is usually determined by the majority class of the training instances that fall into that leaf. For example, if more data points in a leaf are labeled 'Class A' than any other class, that leaf will represent 'Class A'.
- In cases of a tie, different strategies can be employed, such as choosing the class that is most prevalent in the entire dataset.

Measuring Quality of Predictions: Error and Accuracy:

- *Classification Error:* This is a measure of the proportion of instances that are incorrectly classified by the tree. For a given leaf, it is calculated as the fraction of training instances in the leaf that do not belong to the majority class.
- *Classification Accuracy*: This is the complement of the classification error and represents the proportion of instances that are correctly classified. It is calculated as the ration of the number of correctly classified instances to the total number of instances.

Greedy Algorithm to Find the Best Decision Tree:

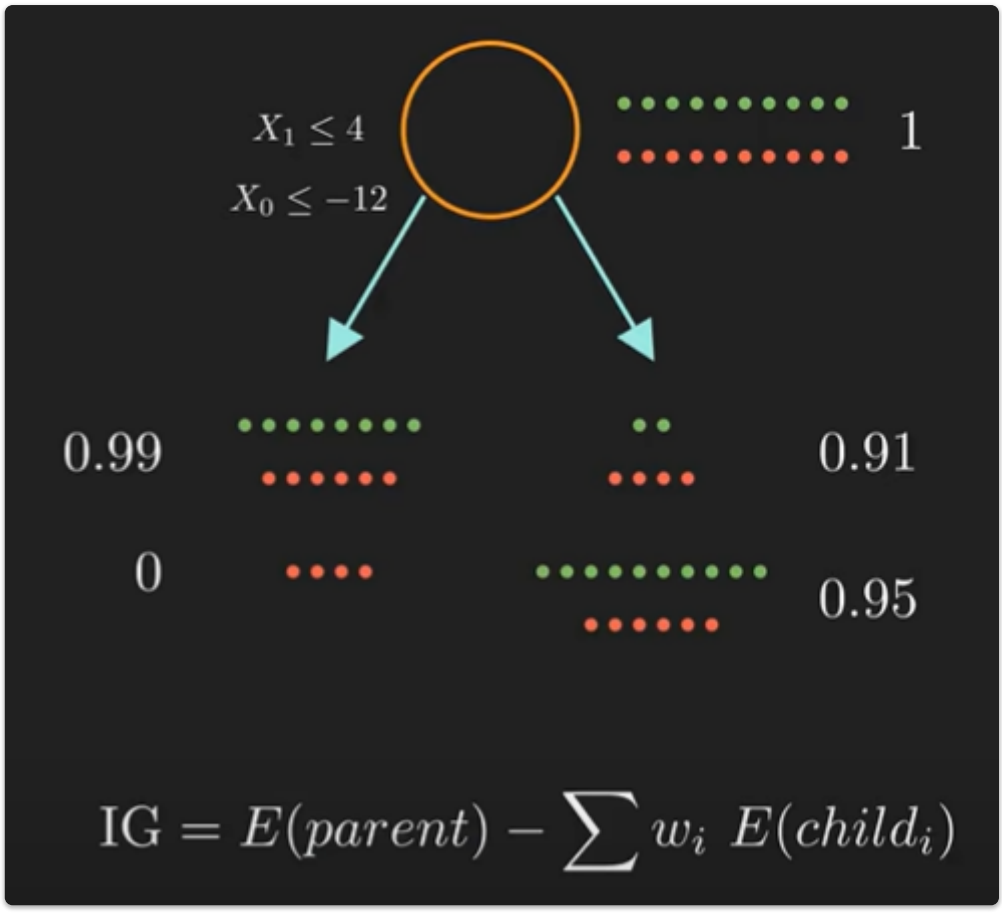
- Decision trees are typically built using a greedy algorithm that makes locally optimal decision at each node. A popular method is the recursive binary splitting.
- The process starts at the root of the tree and splits the data on the feature that results in the largest Information Gain.
- Calculating Information Gain:
  1. Calculate the entropy of each resulting state of the division:

$$Entropy = \sum(-p_i log(p_i))$$

  2. Subtract the combined entropy of all the child nodes from the entropy of the parent node:

$$IG = E(parent) - \sum w_i E(child_i)$$

  where w_i are the weights for the child nodes (relative size of a child with respect to the parent)
  Visual representation of calculating the IG:



- This splitting process is recursively applied to each derived subset. The recursion is completed when the subset at a node all have the same target value or when pre-defined stopping conditions are met (like a maximum depth of the tree or a minimum number of samples in a node).

Measuring Performance of Decision Trees:

- The performance of a decision tree can be evaluated using metrics like accuracy, precision, F1 score, and the area under the ROC curve (AUC-ROC).

- Confusion matrices are also commonly used to visualize the performance, especially to understand the types of errors made by the model.
- Cross-validation is often employed to assess how well the decision tree will generalize to an independent dataset.

5. Greedy decision tree learning: what are the steps for building tree. Stopping conditions for the splitting in the decision tree model. What is the sign of over-fitting in decision trees, how one mitigate this effect: early stopping or pruning. Could you explain what does it mean?

Greedy decision tree learning:

1. *Start at the Root:* Begin with the entire dataset, considering it as the root of the tree.
2. *Choose the Best Split:* At each step, evaluate all possible splits for each feature, and select the one that best separates the data according to a chosen metric (like Gini impurity, information gain or variance reduction for regression trees).
3. *Create Child Nodes:* Split the dataset into subsets based on the chosen split criterion. Each subset forms a new node in the tree.
4. *Recursively Repeat the Process:* Repeat the splitting process for each child node.
5. *Apply Stopping Conditions:* The recursion continues until a stopping condition is met. Stopping conditions can include:
    - All instances in a node belong to the same class (pure node).
    - A maximum depth of the tree has been reached.
    - The number of instances in a node is below a predefined threshold.
    - The improvement in the split criterion (e.g., information gain) is below a certain threshold, indicating little improvement from further splits.

The primary sign of *overfitting* in decision trees is when the tree performs exceptionally well on the training data but poorly on the validation or test data. This discrepancy indicates that the tree has learned the noise and specific details of the training dataset, rather than generalizing from the underlying patterns in the data. Overfitting leads to a decision tree that is overly complex and specific to the training set, making it less effective at predicting new, unseen data.

Mitigating Overfitting:

1. *Early Stopping (Pre-pruning)*:
    - Early stopping involves stopping the tree growth before it perfectly fits the training data.
    - It can be implemented by setting one or more stopping conditions like maximum depth, minimum number of samples per leaf, or minimum improvement in the split criterion.
2. *Pruning (Post-pruning)*:
    - Pruning involves first growing a full tree and then removing nodes from the bottom up if their removal does not significantly harm the model's performance.
    - It can be based on various criteria, such as the error rate, information gain, or cross-validation.
    - Cost-complexity pruning is a common approach where a complexity parameter is used to weigh the trade-off between the tree's size and its accuracy.

6. What are strategies for handling missing data in case of decision trees.

1. **Ignoring Rows with Missing Data:**
    - The simplest approach is to exclude all data points (rows) that contain missing values.
    - This method is straightforward but can lead to a significant loss of data, especially if the dataset isn't large or if missing values are spread across many rows.

2. **Skip features with Missing Data**
3. **Imputation**:
    - Impute missing values with a replacement value. Common strategies for imputation include:
        - Replacing missing values with the mean, median, or mode of the feature.
        - Predicting missing values using other data in the dataset, possibly with another machine learning model.
    - Imputation helps retain the full dataset but can introduce bias, especially if the missing data is not random.
4. **Addapt algorithm:**
    - Explicitly handling missing data by learning algorithm: adding missing value choice to every decision node.

7. Idea of ensemble classifiers and boosting. Could you explain the concept of weighted weak classifiers and weighted data. Could you write down formula for final mode predictions.

Ensemble Classifiers: An ensemble classifier combines multiple individual classifiers to improve the robustness and accuracy of the model. Each individual classifier (like a decision tree, logistic regression, etc.) might be weak when used alone, but when combined, they can provide more accurate and stable predictions.

Boosting: Boosting is a specific type of ensemble technique. It focuses on building a series of weak classifiers and then combining them to form a strong classifier. The main idea is to train subsequent models by focusing on the errors of the previous models. This way, the algorithm pays more attention to the parts of the data that are harder to classify.

Weighted Weak Classifiers: In the context of boosting, each weak classifier is assigned a weight. This weight reflects the classifier's accuracy: more accurate classifiers are given higher weights. The final decision is made based on a weighted vote of all the weak classifiers.

Weighted Data: Boosting algorithms also adjust the weight of the training data. Data points that are misclassified by the current ensemble are given more weight, meaning that subsequent classifiers focus more on these difficult cases.

Formula for Final Model Predictions:
In boosting algorithms like *AdaBoost*, the final model prediction can be represented as a weighted sum of the predictions made by the individual classifiers. Here's a simplified formula:

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$$

where:

- $H(x)$ is the final prediction.
- α_t is the weight of the $t-th$ weak classifier.
- $h_t(x)$ is the prediction made by the $t-th$ weak classifier.
- T is the total number of weak classifiers.
- The $sign$ function returns the sign of the prediction, effectively making the final decision.
  In this formula, α_t is often calculated based on the error rate of the $t-th$ classifier, and $h_t(x)$ represents the individual predictions (usually -1 or +1 in a binary classification problem).
  Each iteration of boosting focuses on the errors of the previous iteration, iteratively improving the ensemble's performance on the training data. The process continues until a specified number of iterations is reached or the error is sufficiently minimized.

8. AdaBoost algorithm, formulas, learning process.

Adaptive Boosting, commonly known as AdaBoost, is an ensemble learning technique that is used for classification problems. It works by combining multiple weak learners to create a strong learner. A weak learner is a classifier that performs only slightly better than random guessing, but by combining several of them in a strategic way, AdaBoost can achieve high accuracy.

Step 1: Initialize the Weights

- All training data points are given an equal and normalized initial weight.

Step 2: Iteratively Train Weak Learners

- For each iteration t in the number of iterations:
  1. *Train a Weak Learner*: A weak learner is trained on the weighted training data. Any algorithm can serve as a weak learner, but decision stumps (one-level decision trees) are commonly used.
  2. *Compute Error*: Calculate the weighted error rate ϵ_t of the weak learner. This is the sum of the weights of the training instances that were misclassified.
  3. *Compute Learner Weight*: Assign a weight α_t to the weak learner's vote. The weight is calculated based on the weak learner's error rate and is given by $\alpha_t = \frac{1}{2} ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$. Learners with lower error rates are given more weight.
  4. *Update Data Weights*: Increase the weights of misclassified training instances so that they are more likely to be correctly classified in the next iteration. Then normalize the weights so that they sum up to 1.

Step 3: Combine Weak Learners

- The final model consists of a weighted majority vote of the weak learners. The output for a new input instance is calculated by summing up the weighted votes from all the weak learners.

Step 4: Make Predictions

- To make a prediction, AdaBoost uses the weighted vote of its weak learners. Each weak learner's prediction is multiplied by its weight and the sign of the total sum determines the final prediction.

Convergence and Overfitting

- AdaBoost can converge to a strong learner as long as the weak learners are better than random. It has been observed that AdaBoost can be resistant to overfitting, especially when the weak learners are simple.

Formula for Updating Weights

- The weights for the next iteration w_i^{t+1} are updated using the formula:

  $$w_i^{(t+1)} = w_i^{(t)} exp(-\alpha_t y_i h_t(x_i))$$

  where y_i is the true label, $h_t(x_i)$ is the prediction of the weak learner, and α_t is the weight of the weak learner.

Clustering & Retrieval

1. Explain TF-IDF representation of documents. What are the metrics which are most commonly used to search for k-NN documents.

Term Frequency (TF):

- *Definition:* Term Frequency measures how frequently a term occurs in a document.

- *Calculation:* It is calculated as the number of times a term t appears in a document d, divided by the total number of terms in the document.

$$TF(t,d) = \frac{Number-of-times-term-t-appears-in-a-document-d}{Total-number-of-terms-in-document-d}$$

- *Purpose:* It gives higher weight to terms that appear more frequently in a specific document.

Inverse Document Frequency (IDF):

- *Definition:* Inverse Document Frequency measures how important a term is within the whole corpus.
- *Calculation:* It is calculated as the logarithm of the number of documents divided by the number of documents that contain the term t.

$$IDF(t,D) = log\left(\frac{Total-number-of-documents-in-corpus-D}{Number-of-documents-with-term-t}\right)$$

- *Purpose:* Terms that appear in many different documents are less significant, so IDF gives lower weight to terms that are common across documents.

TF-IDF:

- *Combination:* TF-IDF is simply the product of TF and IDF:

$$TF\text{-}IDF(t,d,D) = TF(t,d) \times IDF(t,D)$$

- *Use:* This value increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus, which helps to control for the fact that some words are generally more common than others.

Metrics most commonly used to search for k-NN documents:

1. *Cosine Similarity:*
   - Measures the cosine of the angle between two non-zero vectors (here, TF-IDF vectors) of an inner product space.
   - It is particularly useful when the magnitude of the vectors does not matter.
2. *Euclidean Distance:*
   - Represents the 'ordinary' straight-line distance between two points in Euclidean space.
   - Less common in text applications because it is sensitive to the magnitude of the vectors.
3. *Manhattan Distance:*
   - Also known as L1 distance, it computes the sum of the absolute differences of their coordinates.
   - Can be used in high-dimensional spaces but less common in text analytics.
4. *Jaccard Similarity:*
   - Measures similarity between sample sets, defined as the size of the intersection divided by the size of the union of the sample sets.
   - More commonly used for binary or boolean data.
5. *Hamming Distance:*
   - Counts the number of positions at which the corresponding symbols are different.
   - Typically used for binary data.

In practice, cosine similarity is the most popular choice for text data due to its effectiveness in handling the high-dimensional, sparse nature of text data. It is less sensitive to the length of the documents and focuses more on the orientation (or angle) of the document vectors in the vector space.

2. What are the KD-trees. How to build and query KD-tree. What is the complexity of querying and how it compares with complexity of other queries: 1-NN, k-NN.

KD-tree, short for K-dimensional tree, is a space-paritioning data structure used for organizing points in a k-dimensional space. KD-trees are useful in various applications, such as range search and nearest neighbour search, especially in multidimensional keys. They are a binary tree structure that recursively partitions the space into two half-spaces at each node.

Building a KD-Tree:

1. *Start with a Set of Points:* Consider a set of points in k-dimensional space.
2. *Choose an Axis and a Pivot:* At each level of the tree, choose an axis (e.g., x, y, or z in 3D space) and pivot value. Typically, the axis is selected in a round-robin fashion (x, then y, then z, then back to x, etc.) or based on the highest variance. The pivot is often chosen as the median value along that axis among the points to be partitioned.
3. *Partition the Space:* Divide the set of points into two subsets: those with their kth coordinate less then the pivot and those with their kth coordinate greater than or equal to the pivot.
4. *Recursion:* Recursively apply the above steps to each subset, creating left and right child nodes in the tree.
5. *Stop Condition:* The recursion stops when a subset at a node cannot be split further, typically when it has fewer than a certain number of points or when the box hits the minimum width. These nodes become leaf nodes.

Querying a KD-Tree:

1. *Nearest Neighbour Search (1-NN):*
    1. Starting with the root node, the algorithm moves down the tree recursively, in the same way that it would if the search point were being inserted (i.e. it goes left or right depending on whether the point is lesser than or greater than the current node in the split dimension).
    2. Once the algorithm reaches a leaf node, it checks the node point and if the distance is better than the "current best", that node point is saved as the "current best".
    3. The algorithm unwinds the recursion of the tree, performing the following steps at each node:
        1. If the current node is closer than the current best, then it becomes the current best.
        2. The algorithm calculates the distance from our point to the current best and the distance to the section that we chose not to visit.
            1. If the distance to this section is smaller than the distance to the current best, we go to this section.
            2. If it's not, we continue going up the tree.
    4. When the algorithm finishes this process for the root node, then the search is complete.
2. *k-Nearest Neighbours (k-NN):*
    - Similar to 1-NN, but maintain a list of k current best candidates.
    - As a tree is traversed, update the list with closer neighbours.

Complexity:

1. *Building Complexity:* $O(nlogn)$ on average, where n is the number of points. This is due to sorting points at each level of the tree.
2. *Querying Complexity:*
    - 1-NN: $O(logn)$ on average, but can degrade to $O(n)$ in the worst case (e.g., when points are not well distributed).
    - k-NN: The complexity depends on the number of points, k, and the structure of the tree. It's generally $O(logn)$ for well-dsitributed points but can also degrade to $O(n)$.

Comparison with Other Query Methods:

- *Linear Search:* Scanning all points has a complexity of $O(n)$, which is simpler but less efficient for large datasets.
- *Hashing-based Approaches:* Efficient for exact matches ($O(1)$ in the best case), but not suitable for range searches or when an exact match is not required.

- *Other Tree Structures:* Like R-trees or Quad-trees, which are also used for spatial data. The efficiency of these structures depends on the specific use case and the nature of the data.

3. Explain LSH method (locality sensitive hashing). Is it competitive to KD-tree method?

Locality Sensitive Hashing (LSH) is a method for dimensionality reduction with the goal of increasing the efficiency of nearest-neighbor searches in high-dimensional spaces. The core idea of LSH is to hash input items so that similar items map to the same "buckets" with high probability (the hashes are locality-sensitive), while items that are dissimilar map to different buckets.

Principles of LSH

1. *Hashing*: LSH uses hash functions to map high-dimensional data points into buckets of a hash table. These hash functions are designed to increase the probability that similar items will be placed in the same bucket.
2. *Locality Sensitivity*: For any two items, if they are close together (similar) in the high-dimensional space, the probability that they will be hashed to the same bucket is higher than if they are far apart.
3. *Buckets as Candidate Sets*: Items in the same bucket are considered as candidate pairs for being similar or "neighbors."

How LSH Works

1. *Select Hash Functions*: Choose a family of hash functions that are locality-sensitive. For instance, in a vector space, a simple LSH function could be a random hyperplane that divides the space into two parts; the hash function outputs 0 or 1 depending on which side of the hyperplane a point lies.
2. *Hash Data Points*: Apply these hash functions to the data points. Each data point will be hashed several times (once per hash function), and the combination of hash outputs (the hash "vector") will determine the bucket in which the point should be placed.
3. *Create Hash Tables*: The hashing process is usually repeated with different hash functions to create multiple hash tables. This repetition increases the chance that similar points end up in the same bucket in at least one of the hash tables.
4. *Querying*: To find items similar to a query item, hash the query item using the same hash functions and look in corresponding buckets for potential matches. Only those items that fall into the same bucket(s) as the query item are considered for further similarity checks.

Properties of LSH

- *Speed-Up*: By focusing only on the items in the same buckets, LSH can significantly speed up nearest-neighbor searches compared to brute-force search.
- *Approximation*: The result of LSH is an approximation; there's a trade-off between the speed of retrieval and the accuracy of the results. This can be tuned: using more hash functions or tables can improve accuracy but reduces the speed-up.
- *Probability Tuning*: The probability of collision for similar and dissimilar items is controlled by the "radius" parameter in the hash function family. A smaller radius increases the likelihood that only similar items will be hashed to the same bucket.
- *False Positives/Negatives*: LSH may result in false positives (dissimilar items being considered similar) and false negatives (similar items not being recognized). However, these can be mitigated by appropriate choice of hash functions and parameters.

Competitivity of LSH method to KD-tree method

Whether Locality Sensitive Hashing (LSH) is competitive with the KD-tree method depends largely on the context and requirements of the specific problem at hand, particularly the dimensionality of the data and the need for exact versus approximate nearest neighbor searches.

LSH Advantages:

1. *High-Dimensional Data*: LSH tends to be more effective than KD-trees in very high-dimensional spaces. KD-trees suffer from the curse of dimensionality, which makes them less efficient as the number of dimensions grows, sometimes even comparable to a linear search.
2. *Speed for Approximate Results*: LSH is designed for fast retrieval of approximate nearest neighbors, which can be advantageous in applications where speed is critical and exact matches are not necessary.
3. *Scalability*: LSH can handle very large datasets because it uses hash tables, which are generally scalable and efficient for lookup operations.
   KD-Tree Advantages:
4. *Low to Moderate Dimensions*: KD-trees perform well in low to moderate dimensional spaces, where they can be more efficient than LSH for exact nearest neighbor searches.
5. *Exact Nearest Neighbors*: KD-trees provide exact nearest neighbors, which is important for applications where precision is crucial.
6. *Interpretability*: The hierarchical structure of KD-trees can be more interpretable, as it directly corresponds to the structure of the data.

Comparing Performance

- *Dimensionality*: As the dimensionality of the dataset increases, the performance of KD-trees generally degrades much faster than LSH. For high-dimensional data (e.g., hundreds or thousands of dimensions), LSH is often the preferred choice.
- *Query Time*: LSH can offer faster query times for approximate nearest neighbor searches because it avoids examining the entire dataset.
- *Space Complexity*: KD-trees require storage space that grows with the number of dimensions, while LSH requires space for hash tables, which can be more manageable.
- *Accuracy*: KD-trees are typically used for exact matches, while LSH is used for approximate matches. However, the accuracy of LSH can be tuned by adjusting parameters such as the number of hash functions and tables.

LSH is generally more competitive than KD-trees for approximate nearest neighbor searches in high-dimensional spaces and large datasets. For lower-dimensional data or when exact nearest neighbors are required, KD-trees might be the better option.

4. Describe steps of k-means clustering algorithm. How we measure its quality? Could you comment on its convergence.

K-means clustering algorithm:

1. *Choose the Number of Clusters(K)*
   - The first step is to specify the number of clusters - K, you want to divide your data into.
2. *Initialize Centroids (cluster centers)*
   - Randomly select K data points as the initial centroids or generate K random points as centroids.
3. *Assign Points to the Nearest Centroid*
   - For each data point, compute the distance to each centroid and assign the point to the nearest centroid's cluster. The distance is typically Euclidean distance, but other distance metrics can also be used.
4. *Recompute Centroids*
   - Update the centroid of each cluster to be the mean (average) of all points assigned to that cluster.
5. *Iterate*
   - Repeat the steps of assigning points to the nearest centroid and recomputing centroids until one of the stopping criteria is met (see below).
6. *Stopping Criteria:*
   - The centroids have stabilized - there is no change (or minimal change within a threshold) in the centroids between iterations.
   - A predetermined number of iterations is reached.

- The sum of squared distances (or another convergence metric) falls below a certain threshold.

Measuring the Quality of K-Means

The most common measure of cluster quality in K-Means is is the Within-Cluster Sum of Squares (WCSS) which is the sum of squared distances between each point and its corresponding centroid. Lower WCSS indicates tighter clusters. The formula for WCSS:

$$\sum_{j=1}^{k} \sum_{i:z_i=j} ||\mu_j - x_i||_2^2$$

where:

- $\sum_{j=1}^{k}$: Sum over j represents the summation over all clusters k.
- $\sum_{i:z_i=j}$: A conditional sum that adds up only those data points i that have been assigned to the cluster j (indicated by the condition $z_i = j$).
- $||\mu_j - x_i||_2^2$: The squared Euclidean distance between a data point x_i and the centroid μ_j of the cluster to which x_i has been assigned.
  The formula calculates the sum of the squared distances from each point x_i to the centroid μ_j of its assigned cluster. The goal in K-means is to minimize this value, as a smaller WCSS indicates that the data points are closer to their respective centroids, implying better clustering.

Convergence of K-Means

- *Convergence Behavior:* K-means is guaranteed to converge. The algorithm iteratively improves the assignment of data points to clusters and the positions of centroids.
- *Local Optima:* While K-means will converge, it may converge to a local optimum depending on the initial centroid positions. This local optimum may not be the best clustering solution.
- *Random Initialization:* Due to its sensitivity to initial centroid placement, K-means is often run multiple times with different random initializations. The best clustering result (e.g., with lowest WCSS) is chosen.

5. Explain probabilistic approach for clustering. The soft assignment can be optimised with MLE approach (maximum likelihood estimator). Can you explain what does it mean, give some formulas?

A **probabilistic approach to clustering** refers to methods that use probability distributions to model data and infer the structure of clusters. These methods often provide a soft clustering, where instead of assigning each data point to a single cluster, they assign probabilities that a data point belongs to each of the possible clusters. This contrasts with deterministic approaches like k-means, which definitively assign each point to a single cluster.

Some probabilistic approaches to clustering:

1. Gaussian Mixture Models (GMMs)
2. Dirichlet Process (DP) Clustering
3. Quality of Probabilistic Clustering

Using MLE for Soft Assignment

In the context of a Gaussian Mixture Model (GMM), which is a common model for probabilistic clustering, the likelihood of the data given the model parameters is:

$$L(\theta|X) = \prod_{i=1}^{N} \sum_{k=1}^{K} \pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)$$

where:

- $L(\theta|X)$ is the likelihood of the parameters θ given the data X.
- N is the number of data points.
- K is the number of clusters.
- π_k is the mixing coefficient for cluster k, indicating the probability that any point belongs to cluster k.
- $\mathcal{N}(x_i|\mu_k, \Sigma_k)$ is the probability density function of the Gaussian distribution for cluster k with mean μ_k and covariance matrix Σ_k.
- x_i is the i-th data point

Maximizing the likelihood directly is often difficult due to the sum inside the product. Therefore, it's common to maximize the log-likelihood instead, which turns the product into a sum:

$$\log L(\theta|X) = \sum_{i=1}^{N} \log \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k) \right)$$

The MLE process iteratively adjusts the parameters μ_k, Σ_k, and π_k to find the values that maximize the log-likelihood.

Expectation-Maximization (EM) Algorithm

The EM algorithm is a two-step iterative approach to MLE in the context of GMMs:

1. *Expectation Step (E-step)*: Calculate the expected membership probabilities (responsibilities) for each data point for each cluster, which is the probability that each data point was generated by each cluster:

   $$\gamma(z_{ik}) = \frac{\pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)}$$

   where $\gamma(z_{ik})$ is the responsibility that cluster k takes for data point i.
2. *Maximization Step (M-step):* Update the parameters to maximize the log-likelihood based on the computed responsibilities:
   $\mu_k^{new} = \frac{1}{N_k} \sum_{i=1}^{N} \gamma(z_{ik}) x_i$
   $\Sigma_k^{new} = \frac{1}{N_k} \sum_{i=1}^{N} \gamma(z_{ik})(x_i - \mu_k^{new})(x_i - \mu_k^{new})^T$
   $\pi_k^{new} = \frac{N_k}{N}$
   where $N_k = \sum_{i=1}^{N} \gamma(z_{ik})$ is the effective number of points assigned to cluster k.
   The EM algorithm continues alternating between the E-step and M-step until convergence, which is typically determined by a small change in the log-likelihood between iterations.

6. Explain what is the model for "bag-of-words" for clustering documents.

The **bag-of-words (BoW)** model is a simplifying representation used in natural language processing and information retrieval for text analysis. In this model, a text (such as a sentence or a document) is represented as an unordered collection of words, disregarding grammar and even word order but keeping multiplicity.

Step 1: Text Preprocessing

1. *Tokenization*: Split the text into individual words or terms.
2. *Stop Words Removal*: Remove common words that do not contain important meaning and are unlikely to be useful for clustering (e.g., "the", "is", "at").
3. *Stemming/Lemmatization*: Reduce words to their base or root form (e.g., "running" to "run").

Step 2: Constructing the Vocabulary

1. *Vocabulary Creation*: Compile a list of all unique words that appear in the entire corpus of documents. This list of unique words represents the "vocabulary".
2. *Indexing*: Assign an index to each word in the vocabulary.

Step 3: Document Representation

1. *Feature Extraction*: Transform each document into a vector. Each dimension of the vector corresponds to a word in the vocabulary.
2. *Term Frequency (TF)*: Count the number of times each word appears in a document. The value in each dimension of the vector is the count or the term frequency of the corresponding word in the document.
3. *Normalization (Optional)*: Normalize the vectors to prevent bias towards longer documents. This could be done using techniques like TF-IDF (Term Frequency-Inverse Document Frequency).

Step 4: Clustering

1. *Distance Measure*: Choose a distance measure suitable for high-dimensional data, such as cosine similarity, which measures the cosine of the angle between two vectors.
2. *Apply Clustering Algorithm*: Apply a clustering algorithm like k-means, using the vector representations of the documents.

Step 5: Model Evaluation

1. *Cluster Quality*: Evaluate the quality of the clusters, possibly using metrics like within-cluster sum of squares or silhouette score.
2. *Interpretation*: Interpret and label the clusters based on the predominant words in each cluster.

Bag-of-Words Model Characteristics

- *Simplicity*: BoW is simple and computationally efficient, which makes it popular for basic document clustering tasks.
- *High Dimensionality*: The model can result in very high-dimensional vectors (sparse vectors), as many words in the vocabulary may not appear in most documents.
- *Word Frequency*: It considers the frequency of words as features for clustering, which can be good for identifying topics or themes in a collection of documents.
- *Limitations*: The model ignores the context and order of words, which can lead to loss of semantic meaning. Also, synonyms and polysemy (words with multiple meanings) are not handled well.

7. LDA method (Latent Dirichlet allocation). Can you explain the concept?

Latent Dirichlet Allocation (LDA) is a generative probabilistic model used for finding topics in a collection of documents.

Concept of LDA

The fundamental idea behind LDA is that each document can be described by a distribution of topics, and each topic can be described by a distribution of words. LDA tries to backtrack from the documents to find a set of topics that are likely to have generated the collection.

How LDA Works

LDA operates under the following assumptions:

1. *Documents with Similar Topics Use Similar Groups of Words*: LDA posits that if documents share topics, they will use certain groups of words more frequently.
2. *Latent Topics*: Topics are not observed directly but are inferred from the documents and the word frequencies within them.
3. *Dirichlet Distributions*: The distributions of topics in documents and words in topics are assumed to follow Dirichlet distributions, which are controlled by hyperparameters that shape the distributions.

Here's a high-level description of the generative process LDA assumes for a corpus of documents:

1. *For each Topic*:
    - Choose a distribution over words. This is typically a multinomial probability distribution over the entire vocabulary, which defines the probability of each word appearing in a topic.

2. *For each Document*:
    - Choose a distribution over topics. This distribution is also typically a multinomial probability distribution that defines the mix of topics for that document.
    - For each word in the document:
        - First, choose a topic from the document's distribution over topics.
        - Then, choose a word from the corresponding topic's distribution over words.

Parameters of LDA

- *Alpha (α)*: The parameter of the Dirichlet prior on the per-document topic distributions. A higher alpha value encourages each document to contain a more uniform distribution of topics.
- *Beta (β)*: The parameter of the Dirichlet prior on the per-topic word distribution. A higher beta value encourages each topic to contain a more uniform distribution of words.

Inference in LDA

Since the actual process of generating the documents is not known, LDA uses various techniques like variational Bayes and collapsed Gibbs sampling to infer the topic distributions. These techniques aim to maximize the likelihood of the observed documents by adjusting the topic structure.

Applications of LDA

LDA is widely used in natural language processing to model the thematic structure of large collections of documents. It's applied in:

- *Topic Modeling*: To discover abstract topics across various documents.
- *Information Retrieval*: To enhance search by considering topic distributions.
- *Document Classification*: To improve classification by using the topics as features.

8. Hierarchical clustering. Explain algorithm, illustrate with dendrogram.

Hierarchical clustering is a method of cluster analysis which seeks to build a hierarchy of clusters.

There are two main types of hierarchical clustering:

1. *Agglomerative (Bottom-Up)*: This approach starts with each data point as its own cluster and merges them into successively larger clusters.
2. *Divisive (Top-Down)*: This approach starts with all the data points in a single cluster and recursively splits the cluster into smaller clusters.

Agglomerative Hierarchical Clustering Algorithm

1. *Initialization*:
    - Treat each data point as a single cluster. Hence, if there are N data points, you start with N clusters.
2. *Similarity Computation*:
    - Compute the similarity (or distance) between each pair of clusters. Various metrics can be used, such as Euclidean distance for quantitative data or others like Jaccard distance for categorical data.
3. *Cluster Merging*:
    - Find the two clusters that are closest to each other and merge them into a single cluster. Thus, the number of clusters decreases by one.
4. *Update Distance Matrix*:
    - Update the similarity (or distance) matrix to reflect the distance between the new cluster and the original clusters.
5. *Repeat*:
    - Repeat steps 3 and 4 until all data points are clustered into a single cluster of size N.
6. *Termination*:

- The algorithm stops when only a single cluster remains or when a termination condition based on the desired number of clusters or distance threshold is satisfied.