Exam Study notes

* Decision Trees
* Naïve Bayes
* Generalisation and Evaluation
* Linear Regression
* Logistic Regression
* Optimisation and Regularisation
* Support Vector Machines - kernels(Random forest, Linear, poly) – classification bit
* Ethics and Machine Learning
* Nearest Neighbours
* K-Means clustering
* Gaussian Mixture Models
* Principal Components Analysis
* Hierarchical clustering
* Neural networks

**JUST FYI**

**Supervised Learning:**

Linear Regression - Regression

Logistic Regression - Classification

Decision Trees - Classification and Regression

Random Forest - Classification and Regression |

Naive Bayes – Classification

Support Vector Machines - Classification and Regression |

k-Nearest Neighbors (k-NN) - Classification and Regression |

Neural Networks (Multi-Layer Perceptrons) - Classification, Regression, and others depending on the architecture

Gradient Boosting Machines - Classification and Regression (not for the course)

**Unsupervised Learning:**

K-Means Clustering – Clustering |

Hierarchical Clustering – Clustering |

Gaussian Mixture Models – Clustering |

Neural Networks (Autoencoders) - Dimensionality Reduction, Feature Learning, and others depending on the architecture |

Principal Component Analysis (PCA) - Dimensionality Reduction |

t-Distributed Stochastic Neighbor Embedding (t-SNE) – Visualization (not for the course)

Association Rule Learning (Apriori) - Association Rule Mining (not for the course)

Anomaly Detection - Outlier Detection

**Semi-Supervised Learning:**

Self-Training - Classification

Co-Training - Classification

Multi-View Learning - Classification

Transductive SVMs - Classification

Deep Generative Models (Variational Autoencoders) - Dimensionality Reduction, Feature Learning, and others depending on the architecture [(not for the course)]

Note that some algorithms, such as decision trees and neural networks, can be used for both classification and regression tasks depending on the specific problem and the chosen parameters.

**Decision Trees**

Decision trees are a type of supervised learning algorithm, meaning that they require labelled training data to learn from. A decision tree consists of a root node, internal nodes, and leaf nodes. Each internal node represents a decision based on a feature, and each leaf node represents a predicted outcome.

The goal of a decision tree is to create a model that can accurately predict the outcome of new, unseen data. Or The goal of a decision tree is to create a model that predicts the value of a target variable based on several input variables.

*[ When the Decision Tree algorithm is building the decision tree, it tries to pick an attribute to split on where the purity of the child nodes is highest.*

*The purity of the child nodes refers to how much the class labels of the instances in each child node are the same. A good split is one that results in child nodes with high purity, meaning that the instances in each node belong to the same class, or are as close as possible to belonging to the same class. The purity of child nodes is typically measured by metrics such as information gain, gain ratio, or Gini index. The algorithm chooses the attribute that maximizes one of these metrics to make the split]*

To build a decision tree, the algorithm recursively splits the data into subsets based on *the feature that provides the most information gain or reduction in entropy*. Information gain measures how much a particular feature reduces the uncertainty of the outcome, while entropy measures the level of disorder or randomness in the data.

Once the tree is built, the algorithm can use it to predict the outcome of new, unseen data by following the path from the root node to the appropriate leaf node.

The tree is built by recursively splitting the dataset into subsets based on the values of the input variables. The splitting process continues until a stopping criterion is reached, which can be a maximum depth of the tree, a minimum number of samples at a leaf node, or a maximum impurity decrease. The impurity measure is used to determine the best split at each decision node. There are several impurity measures, including entropy, Gini index, and classification error. The formula for each impurity measure is given below:

**Entropy:**

H(S) = - Σ(p\_i log\_2 p\_i)

Gini Index:

G(S) = 1 - Σ(p\_i^2)

Classification Error:

CE(S) = 1 - max(p\_i)

Where:

S is the subset of data at a decision node

p\_i is the proportion of samples that belong to class i in subset S

To build a decision tree, we need to use an algorithm that determines the best split at each decision node. One such algorithm is the ID3 algorithm, which uses the information gain as the criterion for choosing the best split. The information gain measures the reduction in entropy or impurity after a split. The formula for information gain is given below:

**Information Gain:**

IG(S, a) = H(S) - Σ((S\_v/S) \* H(S\_v))

Where:

S is the subset of data at a decision node

a is the feature used for the split

S\_v is the subset of data that has the value v for feature a

H(S) is the entropy of subset S

H(S\_v) is the entropy of subset S\_v

**Advantages and disadvantages of DT**

**However, they can suffer from overfitting and may not perform well on complex problems with a large number of features.**

Advantages:

It is non-parametric (it does not

*Easy to understand and interpret:* Decision trees provide a clear and easy-to-understand graphical representation of the decision-making process, making them useful for explaining and communicating the reasoning behind a decision.

*Can handle both categorical and numerical data:* Decision trees can handle both categorical and numerical data, making them versatile and useful in a wide range of applications.

*Robust to noise:* Decision trees can be robust to noisy data, as they can identify and prune irrelevant features or split points.

*Can handle non-linear relationships*: Decision trees can capture non-linear relationships between features and the target variable, making them useful in modeling complex systems.

*Can be combined with other algorithms*: Decision trees can be used in combination with other algorithms, such as random forests and boosting, to improve performance and accuracy.

Disadvantages:

*Overfitting*: Decision trees can be prone to overfitting, especially if the tree is deep or the dataset is small. This can lead to poor generalization and accuracy on new data.

*Instability*: Decision trees can be unstable, as small changes in the data or model parameters can result in significant changes to the tree structure.

*Biased towards features with more levels:* Decision trees can be biased towards features with more levels or categories, as they tend to provide more information gain than features with fewer levels.

*Limited to binary splits*: Most decision tree algorithms are limited to binary splits, which can result in suboptimal splits and reduced accuracy. [Greedy approach: Decision trees use a greedy approach to recursively partition the data, which can result in suboptimal splits and less accurate models]

*Limited expressiveness*: Decision trees are limited in their expressiveness compared to other machine learning models, such as neural networks, which can model more complex relationships between the features and the target variable.

*Can be sensitive to small changes in the data*: Decision trees can be sensitive to small changes in the data, which can result in significant changes to the tree structure and predictions.

OR

Advantages

Decision trees are easy to understand and interpret, making them a useful tool for explaining complex models to non-technical audiences.

They can handle both categorical and numerical data and are not affected by multicollinearity.

Decision trees can be used for both classification and regression problems.

They can handle interactions between variables and can capture nonlinear relationships.

They can handle missing data, although the approach used depends on the specific decision tree algorithm.

Disadvantages:

Decision trees can be sensitive to small changes in the data and may produce unstable results.

They can easily overfit the data, especially if the tree is too complex, leading to poor generalization to new data.

Decision trees are prone to bias if the training data is not representative of the population.

They are not well-suited for predicting continuous variables with complex patterns.

Decision trees can be computationally expensive to build and optimize, especially for large datasets or deep trees.

**Performance of DT?**

1. *Accuracy***:** This is the proportion of correctly classified instances out of the total number of instances. It is a common metric for classification problems and can be calculated using the formula:

Accuracy = (TP + TN) / (TP + TN + FP + FN)

where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives.

1. *Precision*: This measures the proportion of true positive predictions out of all positive predictions. It is a useful metric when the cost of false positives is high. It can be calculated using the formula:

Precision = TP / (TP + FP)

1. *Recall*: This measures the proportion of true positive predictions out of all actual positive instances. It is a useful metric when the cost of false negatives is high. It can be calculated using the formula:

Recall = TP / (TP + FN)

1. *F1 score*: This is the harmonic mean of precision and recall and is a useful metric when there is an imbalance between the classes. It can be calculated using the formula:

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

1. *Mean Squared Error (MSE):* This is a common metric for regression problems and measures the average of the squared differences between the predicted and actual values. It can be calculated using the formula:

MSE = (1 / n) \* Σi=1 to n (yi - ŷi)^2 (where n is the number of instances, yi is the actual value, and ŷi is the predicted value.)

6. MAE = (1 / n) \* Σi=1 to n |ŷi-yi|

7.R-squared - Calculate the total sum of squares (TSS) as the sum of the squared differences between the actual target values and the mean of the target values.

TSS = sum((y\_actual - mean(y\_actual))^2)

Calculate the residual sum of squares (RSS) as the sum of the squared differences between the predicted target values and the actual target values.

RSS = sum((y\_pred - y\_actual)^2)

Calculate R-squared as 1 - (RSS/TSS).

R-squared = 1 - (RSS/TSS)

**NB: Classification metrics – precision, recall, f1-score, accuracy**

**Regression – MSE, RMSE, MAE, R-squared**

In addition to these metrics, decision trees can also be evaluated using cross-validation techniques, such as k-fold cross-validation, to assess their performance on unseen data[This involves splitting the data into training and testing sets multiple times and evaluating the model on the testing set each time. The average performance across all splits provides a more reliable estimate of the model's performance on new data].

Overall, the choice of performance metric depends on the specific problem and the goals of the analysis.

**How does DT handle missing values?**

* *Simplying ignoring the missing values (works well when the percentage of missing values is small)*
* *Imputing missing values with mean, median, mode or other statistical method.*
* *Treat missing values as a separate category (as a separate category and create a new branch in the decision tree for missing values) works well when missing values represent a significant portion of the data.*
* *Use algorithms specifically designed for handling missing values: Some decision tree algorithms, such as the ID3 algorithm, can handle missing values by incorporating a measure of the uncertainty associated with each split. This allows the algorithm to choose the split that maximizes the information gain while considering the missing values.*

*NB: the approach for dealing with missing values in decision trees depends on the amount and pattern of missing data, as well as the specific decision tree algorithm being used.*

**How does DT handles outliers?**

1. *Pruning:* Decision tree algorithms may include a pruning step that removes branches or nodes that are unlikely to generalize well to new data. This can help to reduce the impact of outliers on the overall tree structure.
2. *Splitting criteria:* Many decision tree algorithms use a measure of impurity, such as Gini impurity or entropy, to determine the best split at each node. These measures are less sensitive to outliers than other measures such as mean or variance.
3. *Ensemble methods:* Ensemble methods, such as random forests or boosting, can help to reduce the impact of outliers by combining multiple decision trees and reducing the influence of any individual tree. Ensemble methods, such as Random Forest and Gradient Boosted Trees, can also be effective in handling outliers by averaging over multiple trees, thereby reducing the impact of outliers on individual trees.
4. *Robust methods:* Some decision tree algorithms, such as CART (Classification and Regression Trees), include built-in methods for handling outliers. For example, CART uses median instead of mean to calculate impurity measures, which is more robust to outliers. Or Minimum Description Length (MDL) tree, which uses a split criterion that is less sensitive to outliers than the standard information gain criterion.

*Overall, the exact approach for handling outliers in decision trees will depend on the specific algorithm and parameters used, as well as the nature and severity of the outliers in the data.*

**How does it handle categorical and numerical data?**

*decision trees can handle both categorical and numerical data. Categorical data is handled by splitting on the different categories, while numerical data is handled by selecting a split point that maximizes the reduction in impurity.*

**Some questions**

How does the concept of entropy relate to decision trees?

*Entropy is a measure of the level of disorder or randomness in the data. In decision trees, entropy is used to measure the homogeneity of a set of examples. The goal of a decision tree is to create subsets of the data that are as homogeneous as possible with respect to the target variable. By splitting the data based on the feature that provides the most reduction in entropy, the algorithm can create subsets that are more homogeneous than the original data.*

**What are some common techniques for preventing decision trees from overfitting to the training data?**

*Pruning:* This involves removing branches or nodes from the decision tree that do not improve the model's accuracy on the validation data. There are two types of pruning: pre-pruning, which stops the tree from growing beyond a certain depth or number of samples per leaf, and post-pruning, which removes nodes from an already grown tree.

*Minimum sample split:* Setting a minimum number of samples required to split a node ensures that only informative splits are made. This prevents the tree from making splits based on noise or outliers.

*Maximum number of leaf nodes*: Setting a maximum number of leaf nodes limits the complexity of the tree, which helps prevent overfitting.

*Maximum depth:* Setting a maximum depth for the tree can prevent it from becoming too complex and overfitting.

*Early stopping*: This involves stopping the tree from growing when the improvement in accuracy on the validation set is not significant.

*Ensemble methods:* Using ensemble methods such as bagging, boosting, or random forests can improve the performance of decision trees by combining multiple trees and reducing overfitting.

Overall, a combination of these techniques can be used to prevent decision trees from overfitting to the training data and to improve their performance on new, unseen data.

Real world Application of DT

decision trees are widely used in various real-world applications in machine learning. Here are a few examples:

*Fraud Detection:* Decision trees are commonly used to detect fraudulent activities in banking and finance. The decision tree can be trained on past data to learn patterns and features that are indicative of fraud. When new data is input, the decision tree can quickly classify whether it is fraudulent or not.

*Customer Segmentation:* Decision trees can also be used to segment customers based on their behavior, demographics, or preferences. By segmenting customers, companies can tailor their marketing strategies to specific groups and increase their customer retention rates.

*Medical Diagnosis:* Decision trees can be used in medical diagnosis systems to classify patient symptoms and provide treatment recommendations. The decision tree can be trained on large datasets of medical records to learn patterns and symptoms that indicate specific diseases or conditions.

*Recommender Systems:* Decision trees can be used in recommender systems to predict which products or services a user is most likely to be interested in based on their past behavior or preferences. The decision tree can be trained on past user interactions with products and services to learn patterns and features that are indicative of user interests.

*Image Recognition:* Decision trees can be used in image recognition systems to classify images into different categories. The decision tree can be trained on large datasets of images to learn patterns and features that are indicative of specific categories, such as objects, animals, or people.

Others:

* In the ID3 algorithm, when considering a node that splits on attribute A, a child node is created for : each value of attribute A occuring in the training data.
* If we prune the tree at level L, then the total number of training instances we find at the leaves (level L, and higher if a set at a node is pure) is We don't know - the particular splits that happen are data dependent, so the number allocated to each node at each level depends on the training data.

Explanation:

The number of training instances allocated to each node at each level in a decision tree depends on the splits that occur, which are data-dependent. Therefore, it is not possible to determine the exact number of instances that will be found at the leaves of a tree at a specific level without knowing the specific splits that occur during the construction of the tree.

* When the Decision Tree algorithm is building the decision tree, it looks at a node and creates a number of child nodes by splitting on an attribute. A good algorithm tries to pick an attribute to split on where the purity of the child nodes is highest (Purer children mean there is more certainty when the classification process reaches them)
* When we are computing purity of a group of instances, can always be measured in bits, which encode the information needed to make a choice between positives and negatives.  If entropy is zero (all positives or all negatives), no further information is needed.  If entropy is one (equal numbers of positives and negatives, so there is no information which to choose), then we need another full bit of information to choose.)
* We use information gain to pick **the attribute to split a node on**.  We pick the attribute with the **the highest information gain**.  However, information gain tends to pick **the attributes with lots of values**.  To avoid this, we can use **the gain ratio**.
* To avoid overfitting, First, obtain **a validation set of instances**.  Then evaluate your tree on this set and remove **the node (and its descendents) which when removed shows the greatest increase in performance on this set**.  Repeat this process until **no further improvement happens no matter which node is removed.**

|  |  |
| --- | --- |
|  |  |

**Naïve Bayes[also here:** [**https://towardsdatascience.com/a-mathematical-explanation-of-naive-bayes-in-5-minutes-44adebcdb5f8**](https://towardsdatascience.com/a-mathematical-explanation-of-naive-bayes-in-5-minutes-44adebcdb5f8) **]**

Naive Bayes is a probabilistic machine learning algorithm that is commonly used for classification problems. It is based on Bayes' theorem, which states that the probability of a hypothesis (such as a particular class label) given some evidence (such as a set of features) is proportional to the probability of the evidence given the hypothesis, multiplied by the prior probability of the hypothesis.

Naive Bayes is considered naive because it strongly assumes conditional independence among features. This assumption means that the presence or absence of one feature has no effect on the presence or absence of another. Due to this simplifying assumption, naïve bayes is computationally efficient and needs relatively small training data when considering practical problems where there are many features, or a high dimensional feature space as compared to using classification models.

The formula for Naive Bayes is as follows:

P(Y | X) = P(X | Y) \* P(Y) / P(X)

where:

* P(Y | X) is the posterior probability of Y given X
* P(X | Y) is the likelihood of X given Y
* P(Y) is the prior probability of Y
* P(X) is the marginal probability of X

The Naive Bayes algorithm assumes that the features are conditionally independent given the class label, which means that the likelihood term can be factorized as follows:

P(X | Y) = P(X\_1 | Y) \* P(X\_2 | Y) \* ... \* P(X\_n | Y) where X\_1, X\_2, ..., X\_n are the n features.

Advantages of Naïve Bayes [ <https://iq.opengenus.org/advantages-and-disadvantages-of-naive-bayes-algorithm/>]

1. It is easy and simple to implement as it requires less computation or training time.

2. It can be used for both binary and multiple class classification related tasks.

3. It is fast and scalable, suitable for large datasets. [It can be used for fast learning and real-time classification tasks, and it can be easily parallelized to run on multiple processors or clusters.]

4. It is simple to understand due to its detailed explanation of classification [ it determines the probability of a specific result and assigns a class based on the highest probability depending the presence or absence of each feature].

5. It is a good choice for small datasets, as it requires less training data due to its feature independence.

6. **Performs well in text classification**: Naive Bayes is a well-liked algorithm for text classification tasks, like sentiment analysis or spam filtering. It performs well. This is because it is capable of handling high-dimensional data and performs well with categorical data, both of which are common in natural language processing.

7.**Robust to irrelevant features**: Naive Bayes is robust to irrelevant features in the dataset. This is because it assumes that all features are independent of each other, and it calculates the probability of a certain outcome based on the presence or absence of each feature independently.

8.**Handles both continuous and discrete data**: Naive Bayes is a versatile algorithm that can be used to analyze a wide variety of datasets because it can handle both continuous and discrete data. Depending on the type of data, it employs various probability distributions, including Gaussian and multinomial

9. **Handles missing data well**: This algorithm is very useful for handling missing data as well. For accuracy measure, this classifier considers only present data and neglect the data which is not present. By this the accuracy is maintained.

Disadvantages

1.**Assumption of independence**:The algorithm makes the assumption that all features are independent of one another, which is frequently false in practical applications. If the features are correlated, this may result in inaccurate classification results.

2.**Lack of flexibility**:Because Naive Bayes is a parametric model, it needs a set of predetermined parameters that must be learned from training data. Its ability to handle complicated and non-linear relationships between features may be constrained as a result.

3.**Data scarcity**:For Naive Bayes to accurately estimate the conditional probabilities of each feature, there must be enough training data. Insufficient training data may cause the algorithm to underperform.

4.**Sensitivity to outliers**:Naive Bayes is sensitive to outliers or extreme values in the data, which can have a significant impact on the estimated probabilities and produce incorrect classification outcomes.

5.**Class imbalance**:When data are unbalanced and one class has significantly more samples than the other, naive Bayes can have trouble handling the situation. This may result in bias in favour of the majority class and suboptimal performance on the part of the minority class.

6.**Limited ability to capture interactions between features**:Naive Bayes may not be able to capture interactions or dependencies between features that are important for classification because it assumes that features are independent of one another.

7.**Limited ability to handle continuous variables**:The Naive Bayes model assumes that the features are discrete or categorical, which prevents it from directly handling continuous variables. The data must be discretized in order to use the algorithm with continuous data, which may cause information loss and decreased performance.

8.**Biased towards features with high frequency**:Biased towards high frequency features: Naive Bayes is susceptible to bias towards features that are prevalent in the training set of data. This could become a problem if some less common but crucial features are missed.

9.**Difficulty in handling missing data**:Naive Bayes has trouble handling missing data, and it does so poorly. The entire instance must be discarded or imputed if a feature has a missing value, which can produce biased results.

10.**Sensitivity to the choice of prior probabilities**:Sensitivity to prior probability selection: Naive Bayes requires prior probability specification for each class, which may have an impact on the classification outcomes. The algorithm's performance may be significantly impacted by the prior probabilities that are selected, which can be arbitrary.

1. **How Naive Bayes handles missing data and their implications:** Naive Bayes handles missing data by ignoring the missing values during training and classification. This means that if a feature value is missing for an instance, that feature is simply not considered when calculating the likelihood of that instance belonging to a particular class. The implication of this is that the accuracy of the Naive Bayes classifier may decrease if there are many missing values or if the missing values contain important information. To handle missing data, some techniques such as imputation can be used to fill in the missing values before training the model.
2. **How Naive Bayes handles outliers:** Naive Bayes is not particularly sensitive to outliers because it relies on the relative frequency of feature values within each class rather than on the absolute values themselves. However, if the outliers are extreme enough to shift the distribution of the feature values within a class significantly, it can affect the classification accuracy.
3. **How the addition of data affects Naive Bayes**: The addition of data to the Naive Bayes model can improve its accuracy if the new data contains useful information that was not present in the original training set. The effect of adding new data on the Naive Bayes model depends on how the new data is related to the existing data. If the new data is very different from the existing data, it may not improve the accuracy and could potentially decrease it.
4. **Types of Naive Bayes and when they are used:** There are three types of Naive Bayes classifiers: Gaussian Naive Bayes, Multinomial Naive Bayes, and Bernoulli Naive Bayes.

* Gaussian Naive Bayes is used for continuous data where the feature values are normally distributed.
* Multinomial Naive Bayes is used for discrete data where the feature values represent counts or frequencies, such as in text classification or spam filtering.
* Bernoulli Naive Bayes is used for binary data where the feature values are either 0 or 1, such as in sentiment analysis or spam filtering.

More notes:

1. Gaussian Naive Bayes: Gaussian Naive Bayes assumes that the features are normally distributed, with the mean and variance of each feature being different for each class. It is commonly used for continuous data.

The Gaussian Naive Bayes classifier uses the following formulas:

* Class prior probability: P(Ci) = n(Ci) / n, where n(Ci) is the number of instances in class Ci, and n is the total number of instances in the training set.
* Likelihood of feature values: P(xj | Ci) = (1 / sqrt(2 \* pi \* varjCi)) \* exp(-(xj - meanjCi)^2 / (2 \* varjCi)), where xj is the value of feature j for the instance, meanjCi and varjCi are the mean and variance of feature j for class Ci, respectively.
* Posterior probability: P(Ci | x) = P(Ci) \* P(x | Ci) / P(x), where P(x) is the evidence probability, and can be computed as the sum of the product of each class prior probability and its corresponding likelihood: P(x) = sum(P(Ci) \* P(x | Ci))

1. Multinomial Naive Bayes: Multinomial Naive Bayes is used for discrete data where the features represent counts or frequencies, such as in text classification or spam filtering.

The Multinomial Naive Bayes classifier uses the following formulas:

* Class prior probability: P(Ci) = n(Ci) / n, where n(Ci) is the number of instances in class Ci, and n is the total number of instances in the training set.
* Likelihood of feature values: P(xj | Ci) = (NjCi + alpha) / (N + alpha \* V), where NjCi is the total count of feature j in class Ci, N is the total count of all features in the training set, V is the total number of unique features in the training set, and alpha is a smoothing hyperparameter.
* Posterior probability: P(Ci | x) = P(Ci) \* product(P(xj | Ci)), where product(P(xj | Ci)) is the product of the likelihoods of all features given class Ci.

1. Bernoulli Naive Bayes: Bernoulli Naive Bayes is used for binary data where the features are either 0 or 1, such as in sentiment analysis or spam filtering.

The Bernoulli Naive Bayes classifier uses the following formulas:

* Class prior probability: P(Ci) = n(Ci) / n, where n(Ci) is the number of instances in class Ci, and n is the total number of instances in the training set.
* Likelihood of feature values: P(xj | Ci) = (NjCi + alpha) / (Nc + 2 \* alpha), where NjCi is the number of instances in class Ci where feature j is present, Nc is the total number of instances in class Ci, and alpha is a smoothing hyperparameter.
* Posterior probability: P(Ci | x) = P(Ci) \* product(P(xj | Ci)^(xj) \* (1 - P(xj | Ci))^(1 - xj)), where product(P(xj | Ci)^(xj) \* (1 - P(xj | Ci))

Others:

* For a given class, it is reasonable to model each of the attributes with its own Gaussian.
* For a given class, it is reasonable to model all the attributes with a general multivariate Gaussian.

Both of these statements are correct. The Gaussian Naive Bayes model assumes that for a given class, it is reasonable to model each of the attributes with its own Gaussian, or alternatively, all the attributes with a general multivariate Gaussian. This assumption simplifies the computation of the class-conditional probabilities by assuming that the attributes are conditionally independent given the class variable. Additionally, the model assumes that the distribution of the attributes for each class is Gaussian or normal.

**Linear Regression**

Linear regression is a statistical method used to analyze the relationship between a dependent variable and one or more independent variables. The goal is to find the best linear equation that can predict the value of the dependent variable based on the values of the independent variables.

The formula for simple linear regression **is *Y = a + bX*,** where Y is the dependent variable, X is the independent variable, a is the intercept, and b is the slope. For multiple linear regression, the formula is ***Y = a + b1X1 + b2X2 + ... + bNXN.***

Pros of linear regression include:

1. It is simple and easy to implement.
2. It can handle both continuous and categorical variables.
3. It can identify the strength and direction of the relationship between variables.
4. It provides a way to make predictions.

Cons of linear regression include:

1. It assumes a linear relationship between variables, which may not always be true.
2. It is sensitive to outliers and can be influenced by extreme values.
3. It can be affected by missing data, which can lead to biased results.
4. It may not perform well when the relationship between variables is complex or nonlinear.

*Handling outliers*: Outliers can have a significant impact on linear regression, as they can skew the results and affect the slope and intercept of the equation. One way to handle outliers is to remove them from the dataset, but this approach may lead to loss of information. Another way is to use robust regression methods that are less affected by outliers, such as the Huber loss function or the least trimmed squares method.

*Handling missing data:* Linear regression assumes that all variables have complete data, but in reality, missing data is common. One way to handle missing data is to remove the rows with missing values, but this approach can result in biased results. Another way is to use imputation methods, such as mean or median imputation, to fill in the missing values.

*Adding data:* When new data is added to the dataset, the linear regression equation needs to be updated to reflect the new relationship between variables. This can be done by re-estimating the slope and intercept of the equation using the new data.

*Performance*: The performance of linear regression can be measured using metrics such as the coefficient of determination (R-squared) or the root mean squared error (RMSE). R-squared measures the proportion of variance in the dependent variable that can be explained by the independent variables, while RMSE measures the average difference between the predicted and actual values of the dependent variable. A higher R-squared and a lower RMSE indicate better performance.

**Logistic Regression**

Note: Generative and Discriminative Classifiers: The most important difference between naive Bayes and logistic regression is that logistic regression is a discriminative classifier while naive Bayes is a generative classifier.

Formula = g(z) = 1/( 1 + e^(−z))

Text, letter

Description automatically generated

Text, letter

Description automatically generated

**Generalisation and Regularisation**

1. Generalisation is about

* how well our classifier performs on unseen data
* how well our classifier performs on testing data
* how well our classifier performs on validation data
* how representative the training data is

Generalisation refers to how well a machine learning model can perform on unseen data. It is evaluated by testing the model on a separate dataset, usually called the testing dataset. The performance of the model on this dataset is a good indicator of its ability to generalise to new data.

To ensure that the testing dataset is representative of the real-world data, it is important to use a validation dataset during the training process. The validation dataset is used to evaluate the performance of the model during training and to make decisions on hyperparameter tuning, model selection, and so on. Therefore, the representativeness of the validation dataset is also important for generalisation.

However, the performance of the model on the training dataset is not a good indicator of its ability to generalise to new data, as the model has already seen this data during training.

2. Overfitting

* A predictor that is too flexible will result in
* A predictor that is too flexible enough will result in
* If predictor A's error rate on training data is less than predictor B's error rate on the same data, but A's error rate on unseen data is greater than B's error rate on the same unseen data, then we say that A is suffering from

Underfitting

* A predictor that cannot capture the relevant patterns in the data results in
* A predictor that is not flexible enough will result in
* If predictor A's error rate on training data is greater than predictor B's error rate on the same data, and A's error rate on unseen data is greater than B's error rate on the same unseen data, then we say that A is suffering from

Just right

If predictor A's error rate on training data is less than predictor B's error rate on the same data, and A's error rate on unseen data is less than B's error rate on the same unseen data, then we say that A is suffering from

3. Models and how their complexities are controlled:

1. Naive Bayes: The number of attributes and limits on the distribution parameters. The model assumes that the features are conditionally independent, which simplifies the computation of the probability distribution over the classes.
2. Decision Trees: The number of nodes and/or the pruning confidence. The complexity of decision trees is controlled by the number of nodes and/or the pruning confidence. Decision trees can be easily overfit to the training data, so controlling their complexity is important to avoid this issue.
3. Linear regression: The degree of the polynomial or the number of attributes. The complexity of linear regression is controlled by the degree of the polynomial, or the number of attributes used to model the relationship between the input features and the output variable.
4. Support vector machines (SVMs): tunning hyperparameters - the cost parameter, kernel etc
5. K-nearest neighbours (k-NN) models – optimising the number of neighbours.

4. *The generalisation error* is the expected error of a model on unseen data, while the training error is the error of the model on the data used to train it. The training error is not a good indicator of the generalisation error, as the model may have overfitted to the training data.

Training error -> build best predictor.

Testing error->estimate generalisation error.

Generalisation error -> cannot compute directly but can estimate

To estimate the generalisation error, we need to evaluate the performance of the model on a separate dataset, usually called the testing dataset. However, this dataset is not available during training, so we need to estimate the testing error from the training error using techniques such as cross-validation or holdout validation. Once we have an estimate of the testing error, we can use it to approximate the generalisation error.

NB:

Estimating the generalisation error from the training error requires us to: can’t be done (because training error does not give us any information about how well the predictor performs with unseen data)

5. If we use N randomly selected testing instances, and get an error rate of E, then the following are true (select all that apply):

E is our best estimate of the error rate in any set of randomly selected instances.

If we use N randomly selected testing instances and get an error rate of E, then E is our best estimate of the error rate in any set of randomly selected instances. This assumes that the testing instances are representative of the population of all possible instances.

However, the error rate on another set of randomly selected instances will not necessarily be E. The variance of the error rate depends on the number of testing instances and the true error rate. It is not distributed approximately Gaussian, but rather follows a binomial distribution.

6. The following options apply to stratification:

* it deals with a problem in estimating errors
* It ensures that testing sets have a representative balance of classes
* It ensures that training sets have a representative balance of classes
* it deals with a problem in k-fold cross-validation
* it is useful for leave-one-out (can be applied in leave-one-out to ensure a representative balance of classes in the testing set)

Stratification is a technique used to ensure that the testing or training sets have a representative balance of classes. It is particularly useful when dealing with imbalanced datasets, where some classes have a much smaller number of instances than others. In stratification, the instances are partitioned into homogeneous groups based on their class labels, and each group is used to form the testing or training set. This helps to ensure that the testing or training set has a similar proportion of instances from each class as the original dataset.

7. We use a confidence interval to describe a range for the testing error rate.

A confidence interval is a statistical measure that tells us how confident we can be that a certain range of values contains the true value of a parameter, such as the error rate of a machine learning model. *When we estimate the error rate of a model on a testing set, we can use a confidence interval to describe the range of error rates we'd expect to see when testing future unseen sets of instances.*

8.

In cross-validation:

* We are less likely to get an unusually biased testing set
* In each fold, the testing set is distinct from the training set
* We pick the subsets in turn, train on the other subsets and test on the one we picked, then average the results.
* Every instance is used for testing
* Every instance is used for training

Leave-one-out is a type of cross validation in which all but one of the instances are used for training

9. (I don’t know)

|  |
| --- |
| Suppose we want to decide whether Naive Bayes or Decision Trees are better for a particular classification task, and for NB which attributes to use, and for DT how many nodes to use, and in the end we want to have an estimate of how the model will generalise to unseen data.  We have *N* labelled instances.  Select which of the following methods could work (select all that apply). |
| |  |  | | --- | --- | | Selected Answers: | We divide the N instances into two sets, L for training and M for testing.  We train  each of the possible models (DT/NB, parameter choices) on L and test each one once on M.  We pick the one that performs best on M and use the error rate on M and the size of M to generate a confidence interval for the error rate on unseen data. | | Answers: | We divide the *N* instances into three sets, K for training, L for validation and M for testing.  We train  each of the possible models (DT/NB, parameter choices) on K and test on L.  We pick the one that performs best on L.  Then we test the chosen model once on M, and use the error rate and size of M to generate a confidence interval for the error rate on unseen data. | |  | We divide the N instances into two sets, L for training and M for testing.  We train  each of the possible models (DT/NB, parameter choices) on L and test each one once on M.  We pick the one that performs best on M and use the error rate on M and the size of M to generate a confidence interval for the error rate on unseen data. | |  | We divide the *N* instances into three sets, K for training, L for validation and M for testing.  We train  each of the possible models (DT/NB) on K and test on L.  We pick the one that performs best on L.  Then we choose the parameters (number of nodes if we picked DT, attributes if we picked NB) by training the model with those parameters on the L and testing once on M. We pick the parameter set that performs best on M, and use the error rate and size of M to generate a confidence interval for the error rate on unseen data. | |  | We divide the *N* instances into three sets, K for training, L for validation and M for testing.  We train  each of the possible models (DT/NB) on K and test on L.  We pick the one that performs best on L.  Then we choose the parameters (number of nodes if we picked DT, attributes if we picked NB) by training the model with those parameters on L and testing several times on M. We pick the parameter set that performs best on average on M, and use the error rate and size of M to generate a confidence interval for the error rate on unseen data. | |

10: and this one too

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | | |
|  | The following are properties of confidence intervals (select all that apply) |  |  |  |
| |  |  | | --- | --- | | Selected Answers: | The confidence interval varies roughly with the square root of the number of samples | |  | It is fully specified by a lower bound and an upper bound | |  | If *N* is reasonably large, then the 95% confidence interval for the error rate in a random sample of size *N* is about two standard deviations either side of the mean | | Answers: | The confidence interval varies roughly linearly with the number of samples | |  | It is fully specified by a confidence level, a mean and an interval either side of the mean | |  | The confidence interval varies roughly with the square root of the number of samples | |  | It is fully specified by a mean and an interval either side of the mean | |  | It is fully specified by a lower bound and an upper bound | |  | The confidence interval varies roughly with the square of the number of samples | |  | If *N* is reasonably large, then the 95% confidence interval for the error rate in a random sample of size *N* is about two standard deviations either side of the mean | |  | If *N* is reasonably large, then the 95% confidence interval for the error rate in a random sample of size *N* is about two standard deviations wide | |  |  |  |

Evaluation

**Hierachical clustering**

Agglomerative

* In **Single Linkage,**the distance between two clusters is the minimum distance between members of the two clusters
* In **Complete Linkage,**the distance between two clusters is the maximum distance between members of the two clusters
* In **Average Linkage,**the distance between two clusters is the average of all distances between members of the two clusters
* In **Centroid Linkage,**the distance between two clusters is is the distance between their centroids

Single agglomerative clustering steps:

1. Begin by treating each instance as a separate cluster.
2. Compute the distance between every pair of clusters using the Euclidean distance matrix. For example, the distance between cluster {a} and cluster {b} is 3, the distance between cluster {d} and cluster {e} is 2, etc.
3. Identify the pair of clusters that are closest together (i.e., have the smallest distance). In this case, the closest pair is {d} and {e}, with a distance of 2.
4. Merge these two clusters into a new cluster, called {d,e}.
5. Recompute the distances between this new cluster and all the other clusters. In this case, the distances are: {a} to {d,e} = 3.9, {b} to {d,e} = 3.2, {c} to {d,e} = 2.5, {f} to {d,e} = 1.
6. Identify the pair of clusters that are closest together. In this case, the closest pair is {f} and {d,e}, with a distance of 1.
7. Merge these two clusters into a new cluster, called {d,e,f}.
8. Recompute the distances between this new cluster and all the other clusters. In this case, the distances are: {a} to {d,e,f} = 4.7, {b} to {d,e,f} = 3.9, {c} to {d,e,f} = 2.7.
9. Identify the pair of clusters that are closest together. In this case, the closest pair is {c} and {d,e,f}, with a distance of 2.7.
10. Merge these two clusters into a new cluster, called {c,d,e,f}.
11. All instances are now in a single cluster, so the algorithm terminates.

Complete agglomerative is the oppsoitve of Single agglomerative. Considers maximum distances . Find examples  
  
  
  
  
Neural Networks ( make comparison for logistic, svm, kneighbours , neural networks and deep neural networks)

**when not to use deep neural networks**

Deep neural networks are not always the best solution, and there are situations where they should not be used. One of the most important disadvantages of deep learning is that it is a black box, meaning that it is difficult to interpret the results and understand how the model arrived at its predictions. Deep learning models also require a large amount of data and computational resources, which can be expensive and time-consuming. In some cases, simpler machine learning algorithms may be more appropriate, especially when the data is limited or the problem is well-defined. Therefore, it is important to carefully consider the problem and the available resources before deciding to use deep neural networks

**KNN (K-Nearest Neighbours)**  
Yes, the dimensionality of the data affects the performance of the k-nearest neighbors (k-NN) algorithm. As the number of dimensions increases, the size of the data space increases, and the amount of data needed to maintain density also increases. This phenomenon is known as the curse of dimensionality, and it affects the performance of k-NN algorithm by making it computationally expensive and less accurate. Without a significant increase in the size of the dataset, k-NN loses all predictive power. Therefore, it is important to consider the dimensionality of the data when using k-NN and to use feature selection techniques to reduce the number of dimensions and improve the performance of the algorithm[**1**](https://www.baeldung.com/cs/k-nearest-neighbors)[**2**](https://towardsdatascience.com/k-nearest-neighbors-and-the-curse-of-dimensionality-e39d10a6105d)[**3**](https://stats.stackexchange.com/questions/159070/curse-of-dimensionality-knn-classifier).

KNN is a simple and intuitive machine learning algorithm that can be used for both classification and regression tasks. The basic idea behind KNN is to use the K-nearest data points in the feature space to a new data point to make a prediction.

Here's how the KNN algorithm works in the case of classification:

1. Choose a value for K, the number of nearest neighbors to consider.
2. Given a new data point, find the K-nearest neighbors in the training set based on some distance metric (e.g., Euclidean distance).
3. Assign the new data point to the class that is most common among its K-nearest neighbors. For example, if K=5 and 3 of the 5 nearest neighbors belong to class A and 2 belong to class B, the new data point is assigned to class A.

And here's how the KNN algorithm works in the case of regression:

1. Choose a value for K, the number of nearest neighbors to consider.
2. Given a new data point, find the K-nearest neighbors in the training set based on some distance metric (e.g., Euclidean distance).
3. Predict the value of the new data point based on the average value of its K-nearest neighbors.

Advantages of KNN:

* KNN is a non-parametric algorithm, which means it does not make any assumptions about the underlying data distribution.
* It can be used for both classification and regression tasks.
* KNN is a lazy algorithm, which means it doesn't need to train a model on the training data. Instead, it stores the training data and uses it for making predictions on new data points.

Disadvantages of KNN:

* KNN can be computationally expensive, especially for large datasets, since it requires calculating the distance between each pair of data points.
* The choice of K can have a big impact on the performance of the algorithm, and it can be difficult to choose a good value of K.
* KNN is sensitive to the scale of the input features, so feature scaling is often necessary.

Preprocessing steps for KNN:

* Normalize or standardize the input features to ensure they are on the same scale.
* Remove any missing values in the dataset or impute them with some value (e.g., mean or median).
* Feature selection can be used to reduce the dimensionality of the dataset and improve the computational efficiency of the algorithm.

Here's the formula for Euclidean distance, one of the most commonly used distance metrics in KNN:

d(x, y) = sqrt(sum((xi - yi)^2)), where x and y are two data points, and xi and yi are their respective feature values.

In summary, KNN is a simple and intuitive machine learning algorithm that can be used for both classification and regression tasks. It has its advantages and disadvantages, and preprocessing steps such as feature scaling and feature selection can be used to improve its performance.

How to find the optimal K

Finding the optimal value of K in the K-nearest neighbors (KNN) algorithm is an important task, as it can greatly impact the performance of the algorithm. There are several methods for selecting the best value of K, including:

1. Grid search: Grid search involves training and evaluating the model for different values of K and selecting the value that gives the best performance on a validation set. The validation set is a portion of the training set that is used for tuning hyperparameters, such as K. The optimal value of K is the one that maximizes the performance metric, such as accuracy or F1-score, on the validation set.
2. Cross-validation: Cross-validation is a more robust method than grid search, as it uses multiple validation sets and averages the performance metrics over them. The most common cross-validation method for KNN is k-fold cross-validation, where the training set is divided into K folds, and KNN is trained and evaluated K times, with each fold being used as the validation set once. The optimal value of K is the one that gives the best average performance over the K validation sets.
3. Elbow method: The elbow method is a heuristic for selecting the optimal value of K based on the sum of squared errors (SSE) or the within-cluster sum of squares (WCSS) of K-means clustering. The idea is to plot the SSE/WCSS against the number of clusters K and look for the "elbow" in the curve, where the SSE/WCSS starts to flatten out. This point is taken as the optimal value of K. The elbow method can also be used for KNN by plotting the performance metric (e.g., accuracy or F1-score) against the value of K and looking for the elbow in the curve.
4. Domain knowledge: The choice of K can also be informed by domain knowledge or prior experience with similar datasets. For example, if the dataset is known to have a certain number of distinct classes, K can be set to that value.

In summary, there are several methods for selecting the optimal value of K in KNN, including grid search, cross-validation, the elbow method, and domain knowledge. The choice of method depends on the dataset and the available resources.

Kinds of data

K-nearest neighbors (KNN) is a flexible algorithm that can work with a variety of data types, including:

1. Continuous data: KNN can handle continuous numerical data, such as height, weight, temperature, etc.
2. Categorical data: KNN can also handle categorical data, such as color, gender, or product type, by converting them into numerical values using techniques such as one-hot encoding or label encoding.
3. Mixed data: KNN can handle datasets that have a mixture of continuous and categorical features by combining the appropriate distance metrics for each type of data.
4. Text data: KNN can be used with text data by converting the text into numerical features using techniques such as bag-of-words or term frequency-inverse document frequency (TF-IDF) encoding.
5. Image data: KNN can be used with image data by converting the images into numerical features using techniques such as color histograms or edge detection algorithms.

In general, KNN works best with datasets that have a small number of features relative to the number of data points, as the computational complexity of the algorithm grows exponentially with the number of features. KNN also works well with datasets that have a clear boundary between the classes, as it relies on the notion of proximity to make predictions. However, if the dataset has noisy or irrelevant features, or if the classes are not well-separated, KNN may not perform as well and other algorithms such as decision trees or support vector machines may be more appropriate.

Dec 2020 IAML past paper, question 4(e)

The clinician asks you to investigate the fairness of the chosen classifier with respect to a protected attribute A. Explain the concept of disparate impact, and describe how you would investigate the performance of the classifier with respect to different values of the attribute A.

Disparate impact is a measure of fairness in machine learning, which is used to detect and prevent discrimination. It is defined as the ratio of the percentage of a protected group that receives a negative outcome to the percentage of the non-protected group that receives the same outcome.

In this case, the protected attribute A could be any variable that is related to the patient's demographic information, such as race, gender, or age. To investigate the performance of the classifier with respect to different values of A, one could first identify the protected group and non-protected group based on the values of A. Then, the classifier can be trained and tested separately on each group to evaluate its performance.

One approach to investigate the fairness of the classifier is to calculate the disparate impact ratio for each group. If the ratio is close to 1, it indicates that the classifier is performing equally well for both groups. However, if the ratio is much higher for one group than the other, it could be an indication of discrimination. In that case, further investigation would be needed to identify the sources of bias and to improve the fairness of the classifier.

SVM

the width of the margin for the maximum-margin hyperplane would change if one of the support vectors is removed.

In a linear SVM, the margin is defined as the distance between the hyperplane and the closest data point from either class. The maximum-margin hyperplane is the hyperplane that maximizes this distance. The support vectors are the data points that are closest to the hyperplane and determine its position.

If we remove one of the support vectors, the position of the hyperplane will change since there will be one less point determining its position. This means that the margin will also change since the closest point to the hyperplane might be different now.

The magnitude of the change in the margin will depend on which support vector is removed and how close it was to the hyperplane. If the removed support vector was very close to the hyperplane, then the change in the margin will be larger than if it was farther away. In extreme cases, removing a support vector might cause the hyperplane to move so much that it is no longer the maximum-margin hyperplane.

Therefore, removing a support vector can have a significant impact on the margin and the hyperplane's position in a linear SVM.

Gauassian Model Mixtures:  
  
Gaussian Mixture Model (GMM) is a probabilistic model used for clustering and density estimation tasks. It is a mixture model where the underlying probability distribution is a linear combination of Gaussian distributions. Each Gaussian distribution represents a cluster, and the weights of the Gaussians indicate the relative importance of each cluster in the overall distribution.

Text

Description automatically generated

Pros:

* GMM can capture complex patterns in the data by modelling the underlying probability distribution as a mixture of Gaussians.
* It is flexible in the number of components that can be used to model the data, and it can handle non-spherical clusters.
* GMM provides a probabilistic framework for clustering, which allows for the estimation of uncertainty in the clustering results.
* It is relatively simple to implement and can be trained efficiently on large datasets.

Cons:

* GMM can be sensitive to the initialization of the model parameters, which can lead to poor convergence to the optimal solution.
* It can also suffer from the problem of overfitting if the number of components is too large.
* GMM can be computationally expensive for high-dimensional data.

Preprocessing steps:

* Standardization of the data is typically recommended to ensure that the covariance matrices are not dominated by any particular feature.
* Outlier removal may be necessary to improve the clustering results.

GMM is used in a variety of applications, such as image segmentation, speech recognition, and anomaly detection. It is also commonly used in mixture discriminant analysis, where it is used to model the class-conditional distributions.

In summary, Gaussian Mixture Model is a flexible and powerful model for clustering and density estimation. It can capture complex patterns in the data and provide a probabilistic framework for clustering. However, it can be sensitive to the initialization of the model parameters and suffer from the problem of overfitting. Proper preprocessing of the data is recommended to improve the clustering results.

AIC and BIC

The AIC (Akaike Information Criterion) and BIC (Bayesian Information Criterion) trade off how well the model fits the data with the complexity of the model.

Both AIC and BIC are model selection criteria that penalize the number of parameters in the model. The basic idea is to find a model that fits the data well but is not too complex, as a model that is too complex may overfit the data and generalize poorly to new data.

The AIC and BIC are based on different principles, but they both balance the goodness of fit of the model (i.e., how well it explains the data) with the complexity of the model (i.e., how many parameters it has). The main difference between the two is the penalty term for the number of parameters.

The AIC uses a penalty that is proportional to the number of parameters in the model, while the BIC uses a penalty that is proportional to the logarithm of the number of data points. This makes the BIC more stringent than the AIC and tends to favor simpler models with fewer parameters.

In summary, the AIC and BIC trade off how well the model fits the data with the complexity of the model by penalizing the number of parameters in the model. The goal is to find a model that is both accurate and simple enough to generalize well to new data.

If we want to select the number of mixtures K in a mixture model, and we are NOT going to use the mixture model in any larger system, then reasonable ways to pick K are:

* The Akaike Information Criterion (AIC): The AIC is a model selection criterion that balances the goodness of fit of the model with its complexity. It provides a measure of the relative quality of different models for a given dataset, and it can be used to select the number of mixtures K that best fits the data. The AIC is defined as AIC = 2k - 2ln(L), where k is the number of parameters in the model and L is the maximum likelihood of the model. Lower AIC values indicate better models.
* The Bayesian Information Criterion (BIC): The BIC is a model selection criterion similar to the AIC, but with a stronger penalty for model complexity. It is defined as BIC = kln(n) - 2ln(L), where n is the sample size and k and L are the same as for the AIC. Like the AIC, lower BIC values indicate better models.
* Splitting the data into a training and validation set and maximizing performance on the validation set: This approach involves randomly splitting the data into two sets: a training set and a validation set. The mixture model is then fit to the training set with different numbers of mixtures K, and the performance of the model is evaluated on the validation set using some metric (e.g., log-likelihood, classification accuracy, etc.). The number of mixtures K that achieves the best performance on the validation set is selected.

Using classifier performance to select K is not a common approach for selecting the number of mixtures in a mixture model, as the mixture model is not typically used as a classifier. Therefore, it is not a reasonable way to pick K if the mixture model is not going to be used in any larger system.

GMM others:  
Expectation Maximisation is a technique to

|  |  |
| --- | --- |
|  | learn parameters for a mixture model, from the data |
|  |  |
| EM is similar to K-Means in the following ways | | |
| |  |  | | --- | --- | | Selected Answers: | neither is able to optimise the number of mixtures/clusters | |  | both reach a local optimum | |  | solution depends on the initialisation | |  | the algorithm terminates when there is insufficient change in the relevant measure | | | |

|  |
| --- |
| If we want to select the number of mixtures *K* in a mixture model, and we are NOT going to use the mixture model in any larger system, then reasonable ways to pick *K* are |
| |  |  | | --- | --- | | Selected Answers: | The Bayesian  Information Criterion | |  | The Akaike Information Criterion | |

|  |
| --- |
| The AIC and BIC trade off |
| |  |  | | --- | --- | | Selected Answer: | how well the model fits the data with the complexity of the model | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | | |
|  | Match up the left and right columns |  |  |  |
| |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  |  | | --- | --- | | Question | Selected Match | | E-step | B.  How likely is it that each data point was generated by each mixture? | | M-step | E.  Estimate the parameters of each of the mixtures, given the probabilities of each data point having been generated by that mixture | | Initialisation | D.  Randomly select mixtures and their likelihoods | | |  |  |  |

|  |
| --- |
| A mixture model is generative in the sense that we assume each data point was generated by **[a]** one of the mixture components, then **[b]** the component's distribution |
| |  |  | | --- | --- | | Selected Answer: | A mixture model is generative in the sense that we assume each data point was generated by **probabilistically picking** one of the mixture components, then **sampling** the component's distribution | |

**K-Means:**

K-means clustering is a popular unsupervised learning algorithm used to cluster data points into K clusters based on their similarity. In this case, we have a one-dimensional dataset with seven instances, and we want to cluster them into K groups. Here's how the K-means clustering algorithm can be used to achieve this:

1. Initialize K cluster centroids: First, we randomly select K points from the dataset as the initial centroids for each cluster. These centroids represent the center of each cluster.
2. Assign data points to clusters: For each data point, we compute the distance to each centroid and assign it to the cluster whose centroid is closest (i.e., has the minimum distance). This step is repeated for all data points.
3. Recalculate centroids: After all data points have been assigned to clusters, we recalculate the centroid for each cluster by taking the mean of all data points in that cluster. This step updates the center of each cluster.
4. Repeat steps 2-3: We repeat steps 2 and 3 until the centroids no longer change or a predefined number of iterations is reached.
5. Final assignment: Once the algorithm converges, we have K clusters with their respective centroids. Each data point is assigned to the cluster with the closest centroid.

The objective function that the K-means algorithm optimizes is the sum of squared distances between each data point and its assigned cluster center:

***J = Σi=1 to n ||xi - μj(xi)||^2***

where n is the total number of data points, xi is the i-th data point, μj is the center of the j-th cluster, and ||.|| represents the Euclidean distance between two points.

In words, the objective function measures how well the data points are clustered around their respective cluster centers. The K-means algorithm seeks to minimize this objective function by iteratively assigning data points to the nearest cluster center, and updating the cluster centers based on the new assignment.

The K-means algorithm is not guaranteed to find the global optimum, as it can converge to a local minimum depending on the initial cluster centers and the data distribution. However, running the algorithm multiple times with different initializations can increase the chances of finding a good solution.

Describe how the K-means algorithm could be adapted to produce a hi- erarchy of clusters, and contrast this with how agglomerative clustering produces a hierarchy.

The K-means algorithm can be adapted to produce a hierarchy of clusters by using a hierarchical approach where the algorithm first splits the data into K clusters using the standard K-means algorithm. Then, each cluster is divided into K sub-clusters using K-means again, and the process is repeated recursively for each sub-cluster until a stopping criterion is met. This approach is known as "top-down" or "divisive" clustering.

On the other hand, agglomerative clustering produces a hierarchy of clusters by starting with each data point as its own cluster and recursively merging the closest pairs of clusters until a stopping criterion is met. This approach is known as "bottom-up" or "agglomerative" clustering.

In summary, the main difference between the two approaches is that K-means clustering starts with a fixed number of clusters and recursively splits them, while agglomerative clustering starts with individual points and recursively merges them into clusters.

IAML 2018

*You have begun a collaboration with an environmental scientist who asks if you can develop a machine learning classifier that can determine if a sample of water is dangerous or safe to drink. As an initial pilot study, you are given a small training dataset which contains information about eight different water samples. Each sample contains a set of values (x1, x2, y), where x1 and x2 are measurements of two different chemical properties of water and y ∈ {0, 1} indicates if the sample is dangerous to drink (i.e. y = 1) or safe to drink (i.e. y = 0). The training set is presented below, where each column is a different water sample:*

*1 2 3 4 5 6 7 8*

*x1. 1.5 2.7 1.7 3.5 1.9 4.0 2.5 0.0*

*x2. 0.0 4.0 3.0 2.0 6.0 5.0 2.0 3.0*

*y. 0 1 1 0 0 0 1 0*

*(a) You have the option to use a Gaussian Naive Bayes, Logistic Regression, or Decision Tree classifier for addressing this problem. For each of the choices, briefly discuss if the classifier, once trained, would result in a training error of 0% for the specific data above. You should justify your answer with reference to the data above, but you do not need to perform the steps required to train the classifiers, nor report the training error for each.*

For Gaussian Naive Bayes, logistic regression, and decision tree classifiers, it is possible to achieve a training error of 0% for the specific data above.

For Gaussian Naive Bayes, this classifier assumes that each feature follows a Gaussian distribution, and it calculates the mean and variance of each feature for each class (safe or dangerous). Based on these estimates, it can then compute the likelihood of observing a specific set of features for each class and use Bayes' rule to calculate the probability of each class given the features. Given that the training set contains only two features (x1 and x2), it is possible to estimate the parameters of the Gaussian distribution accurately, assuming that the distribution of each feature is indeed Gaussian. As a result, the Gaussian Naive Bayes classifier can achieve a training error of 0% on this dataset.

Similarly, logistic regression assumes that the probability of observing a specific class (safe or dangerous) given a set of features is a logistic function of a linear combination of the features. Given that the training set contains only two features (x1 and x2), it is possible to fit a logistic regression model to this data and achieve a training error of 0% if the decision boundary can separate the two classes perfectly.

Finally, a decision tree classifier can also achieve a training error of 0% if it can construct a decision tree that perfectly separates the two classes based on the features. Given that the training set contains only two features (x1 and x2), it is possible to construct a decision tree with a depth of 2 that can perfectly classify the samples in this dataset.

However, it is important to note that achieving a training error of 0% does not necessarily mean that the model will perform well on new, unseen data. Overfitting to the training set is always a possibility, especially with small datasets like this one. Therefore, it is important to evaluate the performance of each classifier on a separate test set before using it to make predictions on new, unseen data.