Artificial Intelligence Pathing

AI has three types Artificial Narrow, General & Super Intelligence

AGI & ASI are human & beyond human level intelligence that in large are only theoretical dreams.

Artificial Narrow Intelligence is specialised intelligence like Machine & in it Deep Learning.
Machine Learning unlike algorithms cannot completely classify and numerate something but they can predict within a reasonable margin of certainty. A definitive algorithm will need all the information of dataset to give its answer making it very big for certain cases and it still wont be able to find answers for questions it does not include in its data set.
Machine Learning will train on the data set to form a form of predicter that is way less in size and hope to get good positive predictions with reasonable accuracy while also being able to process data it has never been introduced to before.

Mathematics required to Clean & Process Data and then Predict Numeration or Classification include Linear Algebra (Matric & Matrix Operations; Scalar, Vectors & Tensors; Singular Value Decomposition; PCA; Determinants & Inverse Matrix; Eigen Values & Vectors; Diagonalisation) Calculus (Deritaves, Partial Derivatives, Chain Rule of Derivations, Gradients, Jacobian and Hessian) Stats & Probability (Covariance & Correlation, Basics of Probability, Expectation, Variance, Bayes Theorem, Random Variables & Probability Distribution Functions, Types of Distributions) and Discrete Mathematics.

Data Collection & Processing : since ANI models are ultimately trained on a specific dataset for whatever answer they have to provide, acquiring data is very important.

Data is found in csv/excel/json files on common ml dataset websites like Kaggle or APIs Requesting Methods like Tweepy or Web Scraping Methods like BeautifulSoup & Scrapy.

Understanding of SQL & NoSQL is important during handling of Data.

This Data is then Cleaned & Processed using Feature Engineering (converting raw information into meaningful usable variables or features), Feature Scaling & Normalisation of the features wrt to all of the data, Reduction of Dimensions to avoid overcomplex & biased models and then finally Feature Selection.

Classical Machine Learning Models rely on feature engineering by humans to process and predict on datasets.

It includes Supervised Machine Learning Techniques where dataset is labelled with its solutions like Regression for Number Prediction(Linear &Polynomial Regression with Ridge, Lasso and ElasticNet Regularisation (adjusting feature influences based on bias, under or overfitting)) and Classification for Classification (KNN, GBM, SVM, Decision Trees, Random Forests & Logistic Regression)

Unsupervised Machine Learning is done on data with no labels where model itself has to find patterns, clusters and associations between them (Clustering, Association Rule Learning & Dimensionality Reduction)

Reinforcement Machine Learning is done via trial & error method by interacting with the environment incorporating concepts like Deep Q Networks, Policy Gradients, Actor Critic Methods & Q Learning

Deep Learning uses complex human brain like Neural Networks to automatically extract features from raw data. Deep Learning Foundations include Perceptrons & Multi Layer Perceptrons, Forward/Backward Propogation, Activation Formulas, Loss functions to implement Deep Learning Architectures like CNN (Convolution, Pooling, Padding, Strides for LeNet, AlexNet, VGG, ResNet, EfficientNet, Faster RCNN), RNN (GRU & LSTM), Attention Mechanism Neural Networks (Self Attention Models, Transformers (BERT & GPTs), MultiHead Attention Models), AutoEncoders and GANs.

NLP is a application of ML and DL to understand and generate human language using concepts like Sentiment Analysis, Named Entity Recognition, Stemming, Tokenisation, Lemmatization, Embeddings & Attention Models.

XAI is set of methods & techniques utilised to interpret and understand what a model is doing because Deep Neural Networks are just blackboxes that can be hard to know whats exactly happening inside its thinking process.

Model Evaluation Methods using metrics like Accuracy, Precision, Recall, F1 Score, ROC-AUG, Log Loss, K-fold Cross Validation, LOOCV & Confusion Matrix usually to evaluate its Generalisation against Bias & Under/OverFitting

Python is a good language for ML because of its well supported ML Libraries.

NumPy is Numerical Library for Efficient Matrix Operations
Pandas for Data Manipulation, Filtering, Grouping, Merging & Normalisation

MatPlotLib for Data Visualisation whereas Seaborn (built on MatPlotLib) for higher levels of Data Analysis and Statistical Plotting.

SciPy for advanced scientific & mathematical computing

Scikit-learn for wide variety of traditional ML model training and Model Evalutions with sklearn.preprocessing + sklearn.pipeline for preparing model preparation and GridSearchCV for rigorously trying all combinations for best hyperparameter tuning while RandomizedSearchCV tries random combinations (hyperparameters are configured settings for models like number of trees, learning rate etc while parameters are just internal variables of models learnt by training on data like weights, vectors & neurons)

PyTorch is research friendly deep learning framework
TensorFlow is production ready deep learning framework

Framework : Django (Enterprise) + Flask (Lightweight) + FastAPI (Rising)

Containers like Docker & Container Orchestration Systems like Kubernetes

AI Agents are an application of Transformer Models like LLM Family (GPTs & BERT) where we provide it with memory, context & well designed prompts (under Prompt Engineering) while giving it access to tools and external APIs & services via frameworks like LangChain to specialise workflows.

Classical Traditional Machine Learning

Features are the measurable properties, attributes, or input variables to represent your data as a set of numbers like age, price, color, gender, texts, images, time.

HyperParameters are higher-level configuration settings defined before training that control how the learning process unfolds like learning rate, number of epochs (training cycles) and number of layers or neurons in a neural network. It doesn't change or depend on the data are set before training and then optimised manually for better performance.

Parameters are the internal variables that a machine learning model learns during training to best fit the data. They are adjusted automatically by optimization algorithms (like gradient descent) and ultimately define the trained model like weights & biases in neural networks, coefficients in linear & logisitic regression and cluster centres in k-means clustering.

Parametric Model assumes a certain fixed structure for your data, like a mold. This structure is controlled by a fixed set of parameters (numbers you estimate from the data).

Imagine you know you want to fit all your data into a straight line (y = mx + b). Your job is just to find the best values for m and b, no matter how much data you have—always just two parameters. This makes the computing very quick and efficient but if the assumption ends up being wrong, then the model performs very poorly.

Examples : Linear Regression, Logistic Regression, Naive Bayes, Linear SVM

Non-Parametric Model don't assume your data follows a certain shape or structure (no fixed "mold"). The number of parameters can grow as you get more data—think of it as being as flexible as your dataset demands as its letting the data speak for itself.

It's a Better Model but Without strong assumptions we need a lot of data to fill in the blanks and they require more computational power and do risk overfitting to noise and its also hard to know why a model has predicted something due to its underlying complexity.

K-Nearest Neighbors (KNN), Decision Trees, Random Forests, Kernel methods like Non Linear SVM with RBF Kernel, Neural Networks.

Bias is the difference between predicted and actual values.

Variance is change in prediction pattern or compactness itself on different datasets

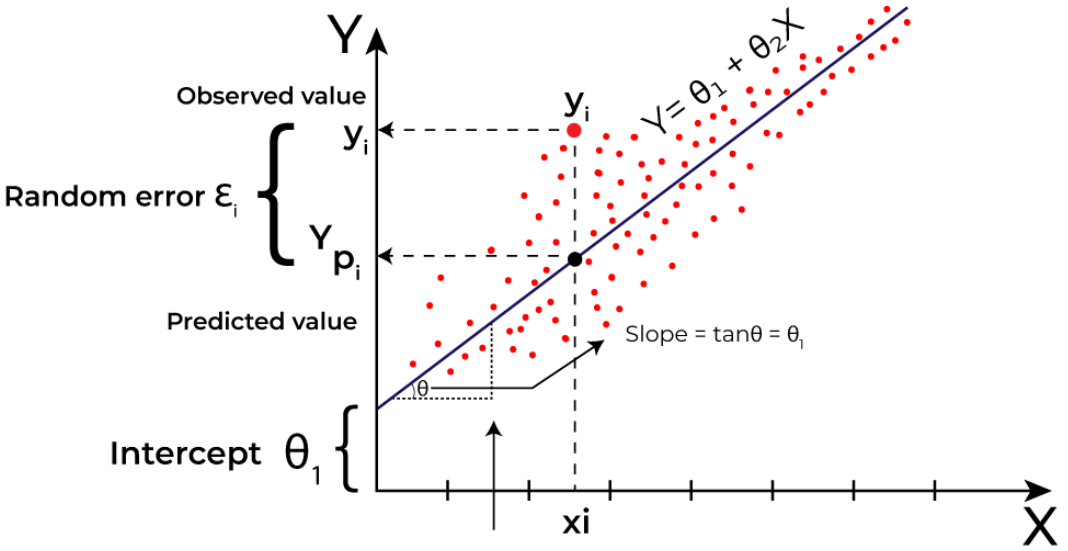
Overfitting Model performs well on training dataset but fails badly on foreign datasets as it's been trained to the point of accommodating even noise in its learning algorithm. **LBHV**

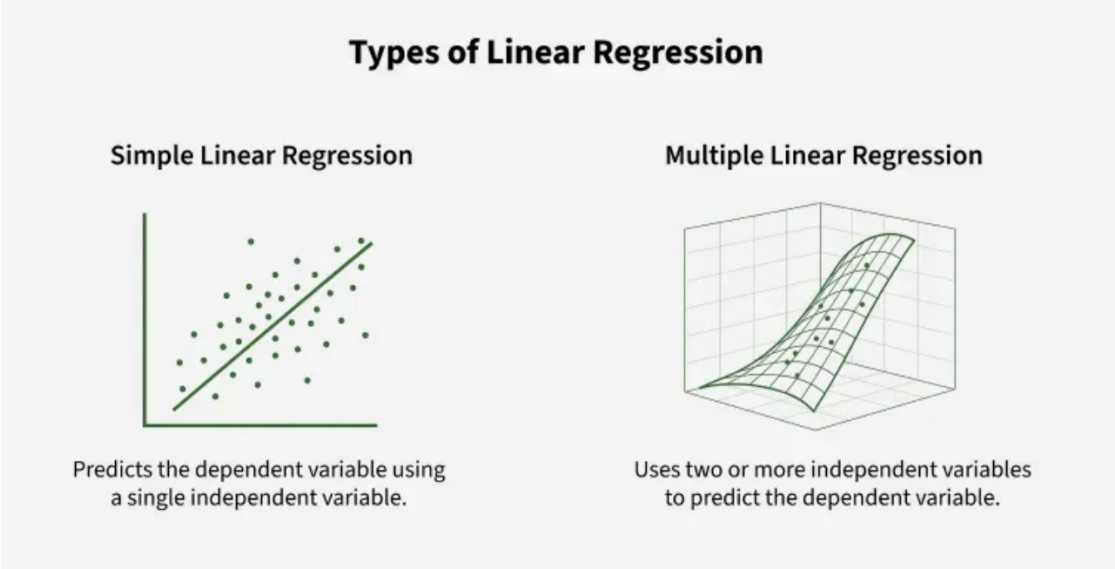
Underfitting Model performs badly both on training dataset and foreign dataset as it doesn't understand the very structure, pattern, relations between the data. **HBLV**

Generalised Model LBLV learn data enough without learning noise to always perform good

Supervised Learning :: Regression

The Model assumes there is a linear relation between inputs and output meaning the output changes at a constant rate as the input changes. A linear equation has an independent variable x and a dependent variable y; if we find a random line that is kind of following the pattern of data and then we find the numerical prediction for a given independent variable; we can just find the corresponding y value on that line because it will most likely be accurate enough.





If we had multiple independent variables deciding one variable then similarly we would have a best-fit plane instead of a best-fit line in which after we input all the independent variables to get the numerical prediction. **Regression returns a numerical value.** Now in

this case the slopes and intercepts are the parameters that can be optimised so that the best fit line has least error & highest predicting performance.
Hypothesis Function h(X) is the equation used to make the regression predictions with parameter values in it for the relationships between input and target value(s)

$$h(x) = \beta_0 + \beta_1 x$$

- $h(x) (or \hat{y})$ is the predicted value of the dependent variable (y).
- x $x$is the independent variable.
- β_0 is the intercept, representing the value of y when x is 0.
- β_1 is the slope, indicating how much y changes for each unit change in x.

$$h(x_1, x_2, ..., x_k) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_k x_k$$

- $x_1, x_2, ..., x_k$ are the independent variables.
- β_0 is the intercept.
- $\beta_1, \beta_2, ..., \beta_k$ are the coefficients, representing the influence of each respective independent variable on the predicted output.

Cost Function (J) is the Mean of Squared Error (Difference between the actual and predicted values). Ideally you want minimum cost function and for that you would need to adjust the parameter values (slope and intercept in this case) to meet those demands.

$$\text{Cost function}(J) = \frac{1}{n} \sum_n^i (\hat{y}_i - y_i)^2$$

Gradient Descent is the method used to minimise the cost function. You start with random parameters for slope and intercept and find the cost function; then you find partial derivatives (partial derivatives are the derivatives of a variable if all other variables are considered constant to check the exact influence of that variable in the equation) of slope and intercept separately to see which parameter has more influence on the error.

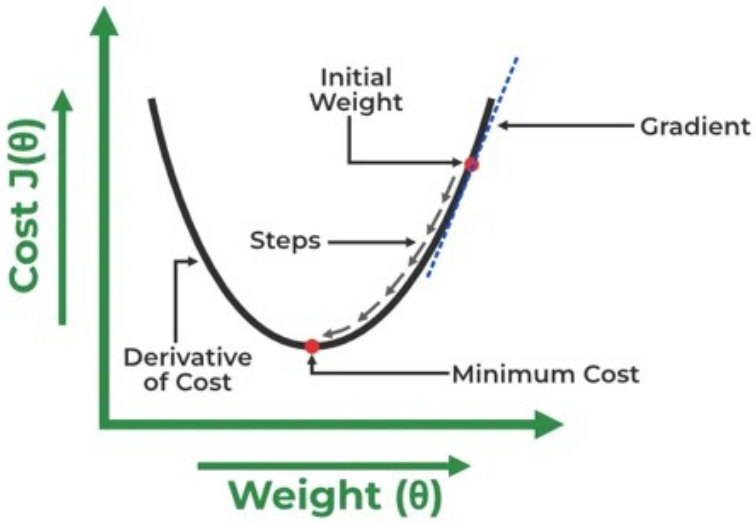
- $\frac{\partial J}{\partial m}$ for the slope
- $\frac{\partial J}{\partial b}$ for the intercept

This gives you the direction and magnitude to change the parameter for minimised error where alpha is the learning rate or the reference size of change in parameters each cycle

- Instead of ONLY changing the parameter that contributes the most to the error, **you update BOTH parameters at the same time**, but by different amounts — proportional to their gradients:

$$m = m - \alpha \cdot \frac{\partial J}{\partial m}$$
$$b = b - \alpha \cdot \frac{\partial J}{\partial b}$$

where α is the learning rate. geeksforgeeks +1



we can use Mean Absolute Error or Root Mean Squared Error as well for Checking Error and Gradient Descents; Obviously MSE & RMSE focus on outliers or drastic errors while MAE gives equal weight to every error.

Regularisation is the process of optimising our parameters better to prevent overfitting by adding a penalty term to normal cost function where the penalty term is :

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(\widehat{y_i} - y_i)^2 + \lambda\sum_{j=1}^{n}|\theta_j|$$

absolute values of coefficients of each feature (slopes) in Lasso Regression (L1)

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(\widehat{y_i} - y_i)^2 + \lambda\sum_{j=1}^{n}\theta_j^2$$

or sum of squares of coefficients of each feature (slopes) in Ridge Regressions (L2)

here in the normal part of cost function instead of 1/n we use 1/2m(where m and n are the same thing) but the ½ is just there because it doesn't affect the minimisation of error process but when we will take partial derivative of each error square wrt to the function, the 2 coming out will get cancelled giving us a neat equation to calculate & lambda is just a regularisation strength term.

The reason we do this is to obtain a new cost function to minimise where we have to reduce large parameters / large coefficients from the equation to minimise the cost function because we are adding them directly or squared now. This is done because large coefficients for features w.r.t to other smaller coefficients in a Model are usually because of overfitting to the noise in a dataset & bring in high variance on new foreign datasets.

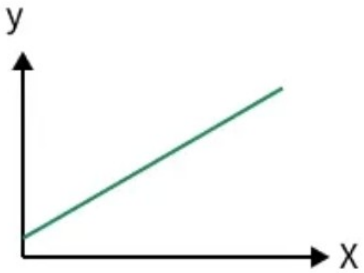
$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(\widehat{y_i} - y_i)^2 + \alpha\lambda\sum_{j=1}^{n}|\theta_j| + \frac{1}{2}(1-\alpha)\lambda\sum_{j=1}n\theta_j^2$$

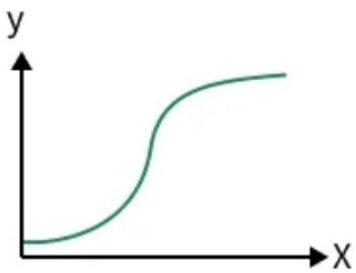
In elastic net regression we just add both penalties and control strength with alpha

Supervised Learning :: Classification :: Logistic Regression

Classification is about accurately predicting the class of an object from given input feature coefficients (slopes) and bias (intercept). In Logistic Regression we assume that there are only two classes so we can classify them as 0 or 1 and there is a linear relation between the log odds and then dependent variable (which in this case is not the final numerated answer but just a term to help us give the final accurate class, in other words a linear predictor).

We also assume the log of odds (probability of it being class 1 / probability of it being class 0) being directly proportional to linear predictor because then we get a squashed function for the relationship between probability of it being class 1 for a given dependent variable or linear predictor. So we can just create a threshold like if probability is more than 0.5 its just class 1 and vice versa instead of having to find a sensible threshold for linear regression where all results are scattered throughout integers, here the scattered probabilities are between zero and one.





$$z = (\sum_{i=1}^{n} w_i x_i) + b$$

The sum of products of feature coefficients into feature variables + bias / intercept gives us linear predictor z and the core assumption of logistic regression is that log of odds is equal to linear predictor / dependent variable of the equation which we can exponentiate and convert to our final sigmoid squashing function.

$$\log\left[\frac{p(x)}{1-p(x)}\right] = z$$

$$
\begin{aligned}
\log\left[\frac{p(x)}{1-p(x)}\right] &= z \\
\log\left[\frac{p(x)}{1-p(x)}\right] &= w \cdot X + b \\
\frac{p(x)}{1-p(x)} &= e^{w \cdot X + b} \quad \cdots \text{Exponentiate both sides} \\
p(x) &= e^{w \cdot X + b} \cdot (1 - p(x)) \\
p(x) &= e^{w \cdot X + b} - e^{w \cdot X + b} \cdot p(x)) \\
p(x) + e^{w \cdot X + b} \cdot p(x)) &= e^{w \cdot X + b} \\
p(x)(1 + e^{w \cdot X + b}) &= e^{w \cdot X + b} \\
p(x) &= \frac{e^{w \cdot X + b}}{1 + e^{w \cdot X + b}}
\end{aligned}
$$

then the final logistic regression equation will be:

$$p(X; b, w) = \frac{e^{w \cdot X + b}}{1 + e^{w \cdot X + b}} = \frac{1}{1 + e^{-w \cdot X + b}}$$

The final logistic regression sigmoid function tells us that probability of the object being class 1 is sigmoid of z or 1 / (1 + e $^{-z}$). The probability then helps us set a threshold for classification like 0.5, the threshold in this case is not a parameter trained by the model but a hyperparameter set before the model training manually by us.

$$L(b, w) = \prod_{i=1}^{n} p(x_i)^{y_i} (1 - p(x_i))^{1 - y_i}$$

Maximum Likelihood function is a equation used for tuning the parameters in logistic regression, y is the true value here so 1 for class 1 and 0 for class 0; in this equation based on true value of class, either half will be irrelevant but we write both halves in the equation of the likelihood function. We are multiplying probabilities of it being class 1 raised to true value if its class 1 or probabilities of class zero raised to (1- true value) which is (1-0) = 1 while the other term becomes irrelevant if its class zero for each data point. The bigger the likelihood function; better the parameters set by the model.

$$
\begin{aligned}
\log(L(b, w)) &= \sum_{i=1}^{n} y_i \log p(x_i) \; + \; (1 - y_i) \log(1 - p(x_i)) \\
&= \sum_{i=1}^{n} y_i \log p(x_i) + \log(1 - p(x_i)) - y_i \log(1 - p(x_i)) \\
&= \sum_{i=1}^{n} \log(1 - p(x_i)) + \sum_{i=1}^{n} y_i \log \frac{p(x_i)}{1 - p(x_i} \\
&= \sum_{i=1}^{n} - \log 1 - e^{-(w \cdot x_i + b)} + \sum_{i=1}^{n} y_i (w \cdot x_i + b) \\
&= \sum_{i=1}^{n} - \log 1 + e^{w \cdot x_i + b} + \sum_{i=1}^{n} y_i (w \cdot x_i + b)
\end{aligned}
$$

Since multiplying very small numbers can cause errors especially for computers; we natural log both sides and equate to a more easier Log Likelihood function that we want to maximise for good parameter tuning.

Then we can take partial derivatives of each feature variable while keeping all other variables constant to check their influence. If partial derivative of a variable is positive then increasing its coefficient will increase it's the log likelihood and if its negative then decreasing it will increase the log likelihood giving us the direction of change and the size of partial derivative gives us the influence; big partial derivative means even a small change can have drastic effect on the likelihood function size.

$$\begin{aligned}\frac{\partial J(l(b,w)}{\partial w_j} &= -\sum_{i=n}^{n} \frac{1}{1+e^{w\cdot x_i+b}} e^{w\cdot x_i+b} x_{ij} + \sum_{i=1}^{n} y_i x_{ij} \\ &= -\sum_{i=n}^{n} p(x_i;b,w) x_{ij} + \sum_{i=1}^{n} y_i x_{ij} \\ &= \sum_{i=n}^{n} (y_i - p(x_i;b,w)) x_{ij}\end{aligned}$$

Supervised Learning :: Classification :: Decision Trees :: ID3

Iterative Dichomiser 3

Suppose we have four features like age, income, student or not, credit score to decide whether they buy a computer or not and we have a sample data set to train on like

age	income	student	Credit rating	Buys computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31...40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31...40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31...40	medium	no	excellent	yes
31...40	high	yes	fair	yes
>40	medium	no	excellent	no

we would need to find out what feature to choose as the root node for the first splitting and for that we would have to calculate information gain for each feature

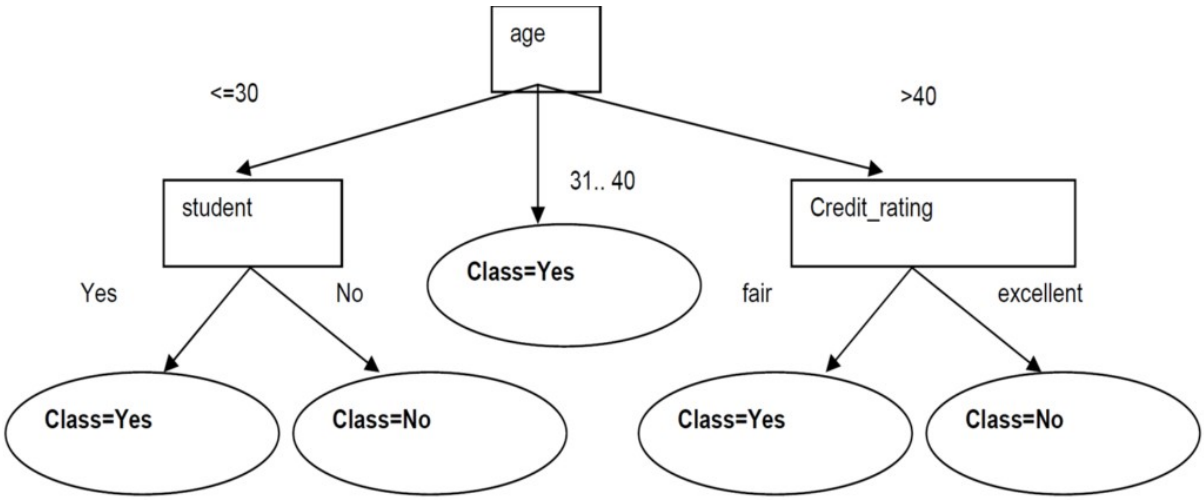
Information Gain = Entropy of Whole Data – Entropy of Each Feature Value

If the Final Output is N different Answers (here 2 answers; yes and no) and we have to calculate what answer will be come in our algorithm for a given dataset then **Entropy of Whole Data** is just :

(– number of final output 1/ total final outputs in data) x $\log_2$(– number of final output 1/ total final outputs in data) + (– number of final output 2/ total final outputs in data) x $\log_2$(– number of final output 2/ total final outputs in data) and so on.
Lets say we wanna calculate Info Gain for income to see if income is the first root node splitter; we already have entropy of whole data set now we need to know the entropy of income which has low, medium and high. we isolate every feature value like low comes 4 times with 3 yesses and 1 no; so the entropy will just be (-3/4)$\log_2$3/4 + (-1/4)$\log_2$1/4 using the same formula but for a smaller table because its just entropy of low income. We calculate the entropy for high and medium income and subtract the total sum of feature value entropy from entropy of dataset to get the info gain.

The Feature with smallest Info Gain is chosen as Root Node Splitter. After we split the Tree intro all pathways of each feature value; we assume its somewhat constant and doesn't affect the further pathway calculations. Now in each pathway we calculate the Info Gain of each remaining feature only taking in the table where initial pathway decision holds true. So if income had highest info gain in first step and we are in income = high path then we only take in the data where income = high to calculate further splitting entropy calculations.

If All Feature values are the same final output value then we consider it a leaf node and end the tree there and calculate for other decision pathways.



This is the answer for the question above and clearly as we can see that age had the Lowest Info Gain in step one and age 31 to 40 had all final outputs as "yes" so we just ended the tree there as a leaf node.

:: CART (Classification & Regression Tree)

CART is somewhat similar but instead of Info Gain we calculate Gini Impurity. In CART we can only do binary splitting that is only two pathways each decision. So for features with two values we can just calculate gini impurity one split possible but for three values we have to calculate gini impurity for all three splitting possibilities like age we have (<=30,31..40 | >40) , (<=30 | 31..40 ,>40) and (<=30 ,>40 | 31..40) and out of all gini impurities we choose the one with least gini impurity -> ignore this feature and only take the table where we have the data chosen in the pathway. Lets say we split (<=30 | 31..40 ,>40) then in one pathway we only take the data of <=30 and now ignore age feature and calculate further until leaf node and in another pathway we take in all data of 31..40 ,>40 and calculate further ignoring this feature now onwards.

Gini Impurity is 1 – (probability of output1)^2 - (probability of output2)^2 and so on.

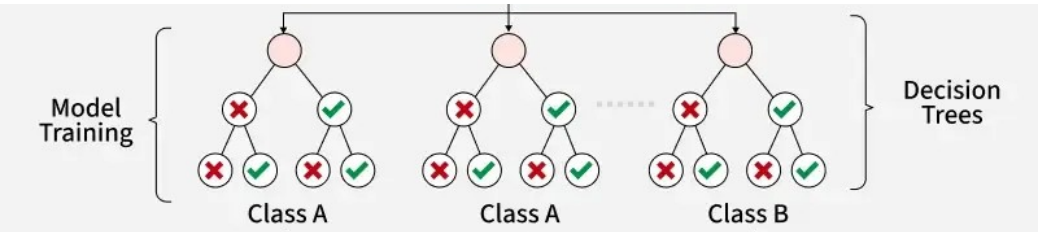
Gini Impurity for whole data set would just be in this example 1 – (9/14)^2 – (5/14)^2 and Gini Impurity of each Binary Split would be Gini Left + Gini Right so we just take in the table where left or right is true (ignore other columns) and just see the 1 – sum of all output probability squares and then choose the split with Least Splitting Impurity Total

```
Age (Split: {31...40} vs. {<=30, >40})
├── 31...40: Buys Computer = Yes (4/0)
│    [4 Samples, 100% Yes]
│
└── <=30 or >40: (5 Yes, 5 No)
    ├── Student = yes: (4 Yes, 1 No)
    │   │   [Split: {yes} vs. {no}]
    │   ├── Credit Rating = fair: Buys Computer = Yes (4/0)
    │   │   │   [Split: {fair} vs. {excellent}]
    │   │   └── Credit Rating = excellent: Buys Computer = No (0/1)
    │   │
    └── Student = no: (1 Yes, 4 No)
        ├── Income = high: Buys Computer = No (0/2)
        │   │   [Split: {high} vs. {medium}]
        │   └── Income = medium:
        │       ├── Credit Rating = fair: Buys Computer = Yes (1/0)
        │       │   [Split: {fair} vs. {excellent}]
        │       └── Credit Rating = excellent: Buys Computer = No (0/1)
        │
```

:: Random Forests

In Random Forests we bootstrap various sections of data sometimes collecting twice or more times and sometimes not collecting at all and create many variants of the dataset. Then Randomly select features for the root node splitting for each tree and then running CART on each decision tree. For a new input you run all your trees and let the majority vote be the final classifier.

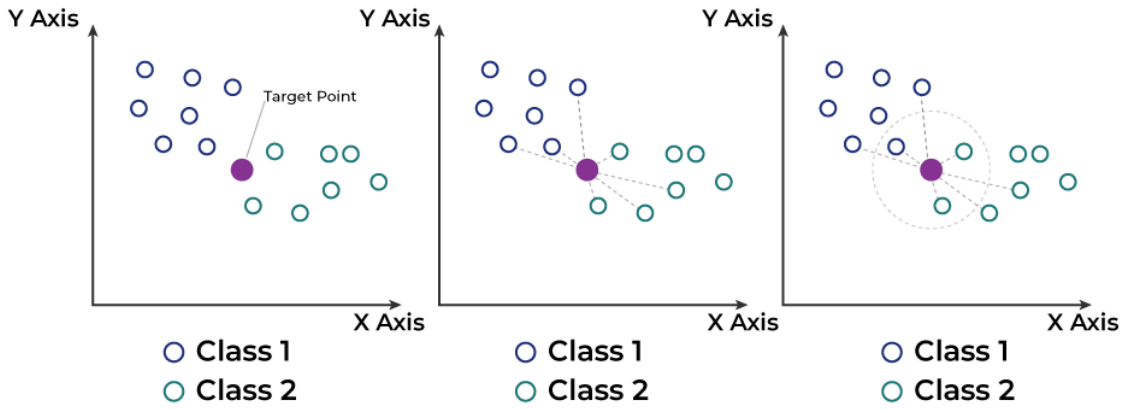
This method prevents overfitting because individual trees can memorise noise and be unstable, with random forests we are no longer sensitive to outliers in data because random feature and data selection ensures each tree has different rules and capture more subtle patterns in the data.



Supervised Learning :: Classification :: K Nearest Neighbours

K- Nearest Neighbours is a non-parametric machine learning algorithm sometimes also called a lazy algorithm because it stores the whole data set and does not really create some sort of weight or parameter for quick predictions.

K is a hyperparameter designating the number of neighbours you check in KNN; for a given set of input features we plot the point on the dataset and find the K closest neighbours either via euclidean absolute distance or manhattan sum of x and y distance and let the majority prediction be the final solution to the input.



AFTER SVM AND GBM SEE ABOUT FEATURE SELECTION EXTRACTION AND DATA CLEANING TECHNIQUES BEFORE GOING TO DL AND SEE THE TESTING METHODS AS WELL SEE WHERE PCA / SVD COME IN AS WELL / AND ALL OTHER MATHS

https://www.perplexity.ai/search/artificial-intelligence-pathin-BhYE.q4FTCCgAQT.xvqg0Q?login-source=sharedThreadLoginGate&login-new=false