# Definitions

* Supervised Learning: Using data labels, we predict outputs using inputted data and labels
* Classification: Decide target class using set of features
  + Binary Classification: Only two target classes
  + Multiclass classification: More than two target classes
  + Eager Learning Classification strategy: Build a model during training phase. Less work at run-time but more offline setup work
  + Lazy Learning Classification Strategy: The classifier keeps all the training data to use later. Less offline work but waits for new query examples
* Regression: Using set of features, determine value of a continuous output variable
* Distance-based models: Via exploitation of distance between two points (generalised training and unseen data), create a model to classify points
* Unsupervised Learning: Unlabelled data inputs are mapped to outputs using data exploration and knowledge discovery algorithms
* Target class label: Value in which our model will output. i.e. what we aim to find
* Feature space: D-dimensional coordinate space that represents inputs with each co-ordinate representing a feature
* Distance function: Measures how close two points in a feature space are
  + Local distance function: Based off only a single feature
  + Global distance function: Based off all features in a feature space
  + Overlap function: Function which returns 0 if two feature values are equals, else 1 (Categorical)
  + Hamming Distance: Sum of overlap function across all features
  + Absolute Difference: Distance between two points then absolute it. For ordinal values, use positions in list as distance
  + Euclidean distance: Square root of sum of squared differences across all featuresA math equation with square root and square root

    AI-generated content may be incorrect.
* Normalisation: The process of making data in a similar range
  + Min-max Normalisation: Rescale values to be in range [0.1] by doing

# KNN

* Nearest neighbour rule (1NN): find a single labelled example close to the input and assign it the same label as x
* K-Nearest Neighbours: Using k-nearest neighbours from training set, assign a label to an input
* Majority voting: Decide on a label using votes of its k nearest neighbours, the majority it the new label
* Here we mainly use distance to determine what is the nearest neighbour and then assign the label
* We can add weights to some of the training data to influence the votes
* Inverse distance weighting: Sum of the inverse distance between point and neighbour
* 1NN susceptible to noise
* Hyperparameter tuning is adjusting settings of an algorithm to better suit data
* In this case, setting k too low leads to easier misclassification but if k is too high, classes become unbalanced
* Process in python:
  + Load csv
  + Set the features and targets (can normalise or add weights)
  + Set up classifier and then train it
  + Make predictions

# Decision Trees

* Idea of a decision tree is to split data into subsets using rules on features which are inferred from training set
* Eager Learning Strategy
* Python implementation:
  + from sklearn.tree import DecisionTreeClassifier,  
    df = pd.read\_csv(df.csv')  
    y = df.pop(target\_feature).values  
    df.pop(Categorical Features)   
    df\_features = df.columns  
    X = df.values  
    tree = DecisionTreeClassifier(criterion='entropy') #Different criterion exist  
    df\_tree = tree.fit(X, y)
* How to create a decision tree:
  + Place all training set into a root node
  + Use one of the features to split data into child nodes, ideally with as many child nodes which don’t split up further
  + Repeat the above but using a different feature on split data and is done until all the initial dataset is split into subsets in which all datapoints are of the same class (i.e. can’t be split further)
* Pure Node: All nodes have same class label (aim is to get this)
* A decision tree in which all the leaf nodes are pure can always be constructed provided there are no clashes in the data
* Aim for simpler trees rather than complex ones and aim for fewer assumptions
  + Good trees classify data using as few tree nodes as possible (minimise depth)
  + Aim for high purity when splitting (go for splitting that leads to highest purity)
  + Use a good feature to divide examples into single category classes
* Entropy: Measure of uncertainty around information source. Low for predictable, high for random
  + Low entropy= high purity
  + High entropy= high impurity
  + Formula: where pj is probability of Class Cj
    - Example: A math equations on a white background

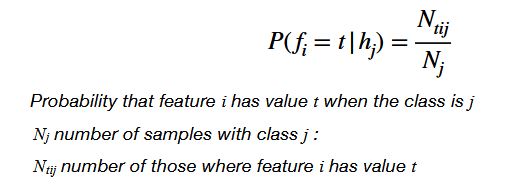
      AI-generated content may be incorrect.
* ID3:
  + Algorithm which builds a tree top down
  + Steps:
    - Start with an empty tree and a set of all training sets
    - If all examples in S are of the same class, then return a new leaf node and give it a label.
    - Else: Select a feature then generate leaf nodes where the selected feature is test feature. Repeat the if else conditions with create subtree
* Information Gain(IG): An approach to select features in decision trees using entropy.
  + Formula: original entropy – entropy after split (note entropy after split will be weighted inn proportion to size) 🡪IG(S,A) = where S is original set and Si is subset
  + Higher IG, the better the selected feature is
* Once tree is constructed, we can use it classify objects
* If we have two identical data (i.e. same features) but different target labels, we can either take the majority vote at a leaf node or randomly choose 1 if a tie
* Overfitting can occur and to avoid this we reduce the models capacity through various means such as setting max depth
* Note, we must convert any categorical variables into numbers, one hot encoding helps with this or we can use get\_dummies from pandas
  + One hot encoding:   
    from sklearn.preprocessing import OneHotEncoder  
    onehot\_encoder = OneHotEncoder(sparse=False)  
    dfOH = onehot\_encoder.fit\_transform(df)
  + Dummies:  
    df = pd.get\_dummies(df,drop\_first=True)

# Naïve Bayes

* Eager learner
* A mathematical equation with black text

  AI-generated content may be incorrect.
  + P(A|B) is conditional probability of A given B
  + P(B|A) is likelihood
  + P(A|B) is known as posterior probability
  + P(A) is usually the probability of hypothesis but its also the probability of encountering a class
* The below table tells us where to find the value for probability calculations

|  |  |  |  |
| --- | --- | --- | --- |
|  | C | D | Total |
| A | P(A|C) | P(A|D) | P(A) |
| B | P(B|C) | P(B|D) | P(B) |
| Total | P(C) | P(D) | P(C) + P(D) = P(A)+P(D) |

* Idea is to use estimates of likelihoods to determine most likely prediction for classification
* Can be used for classification of texts
* As we aim to find the most likely hypothesis for data, we usually use Maximum Aposteriori Hypnothesis (MAP): hMAP  = arg max h∈H P(h|D)
* As it is tough to estimate the likelihood, we use a naïve assumption in that given all the features, our likelihood is where fi is a given label and hj is given feature then take the hj which has the highest probability and multiply it by the likelihood
* Steps:
  + Calculate class prior probabilities
  + Get conditional probabilities for each feature via:  
    
  + Construct a full contingency table for all features on both classes
  + Test a new example by multiplying class feature probabilities by the target class probability. In this case, look at the test input features and multiply the probabilities which has the same features
  + Normalise data
* If we have numerical values as a feature, we can either convert it to take a fixed value or we can use some distribution if it fits some distribution such as storing mean and standard deviation if it fits some norma distribution
  + Normal PDF: A mathematical equation with numbers and symbols

    AI-generated content may be incorrect.
* IN python:
  + CategoricalNB will work with categorical data once it is processed  
    using an OrdinalEncoder
  + GaussianNB assumes the numeric features have a Gaussian  
    distribution
  + BernoulliNB binary data (use with one hot encoding)
  + MultinomialNB count data such as word counts
  + We use OrdinalEncoder to convert values to numbers

# Evaluation

* Null Hypothesis(H0): Assumption being tests. We either reject or fail to reject it.
* Alternative Hypothesis(Ha): Hypothesis we accept if we have enough proof to reject null hypothesis
* Type I error: Reject H0 when it is true. False Positive
* Type II error: Fail to reject null hypothesis when it should be rejected. False negative
* Aim is minimizing False negatives as much as we can. While we want to reduce false positives, a false negative is more serious (think pandemic)
* Misclassification rate = (# incorrect predictions)/total predictions
* Accuracy = (#correct predictions)/total predictions
* Usually, split data into training and test (split is your decision) and train on the training data then use held out data as test data to test accuracy
* Confusion matrix: Summarise performance of model

|  |  |
| --- | --- |
| True Negative(TN) | False Positive(FP) |
| False Negative(FN) | True Positive(TP) |

* Isnt necessarily a 2x2 table, can be multiclass
* Recall(TP rate)=TP/(TP+FN)
* Precision=TP/(TP+FP)
* Specificity(TN Rate)= TN/(TN+FP)
* False Positive Rate)=FP/(FP+TN)
* Can plot precision and recall on a curve to study output
* Imbalanced data: Refers to a problem in classification where the  
  classes in the data are not represented equally
* To evaluate skewed classes:
  + BAR(Balance Accuracy Rate): Mean of TP Rate and TN rate
  + BER(Balance error rate): Mean of FP rate and FN rate
* F-Measure: Measure which trades off precision against recall for given balance
  + Formula: A close-up of a sign

    AI-generated content may be incorrect.
  + Beta parameter is what controls the trade-off. If one, trats both equally, more than one focuses on recall while less than one focuses on precision
  + F1-measure is when beta is one and most widely used
  + Python example:   
    from sklearn.metrics import f1\_score  
    kNN = KNeighborsClassifier(n\_neighbors=3)  
    dtree = DecisionTreeClassifier(criterion='entropy')  
    mnb = GaussianNB()  
    models = [mnb,kNN,dtree]  
    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,  
    random\_state=0,  
    test\_size=1/3)  
    print('F1 Measure')  
    for m in models:  
     mm = m.fit(X\_train, y\_train)  
     y\_pred = mm.predict(X\_test)  
     f1s = f1\_score(y\_test, y\_pred, pos\_label='good')  
     print("F1 Score Test set {:22} {:.2f}".format(type(m).\_\_name\_\_, f1s))
* F1 usually rewards balance between TP rate and TN rate but punishes extreme values
* Good BAR score needs good TP rates
* Use sklearns classification report for reports
* Decision Threshold: Value of theta which discriminates the selection of +ve and -ve outcomes. Usually 0.5
* ROC curve: plot of how TP rates and FP rates change over different thresholds
* The better the ROC curve, the closer it is to the top left corner, meaning a larger area under the graph
* Overfitting: When the model is fitted too close to training the data, it no longer is generalised and gives less accurate outputs
* To prevent overfitting, we can use the training-test split or use cross-fold validation
* K-fold cross validation:
  + Divide data into k disjointed sets
  + Select one fold for test and the rest for training
  + Repeat for all folds then average the accuract/error
* Hold out:
  + Split data into test, training and validation set(used to tune classifier)
  + Can use k fold with this
  + Is slower