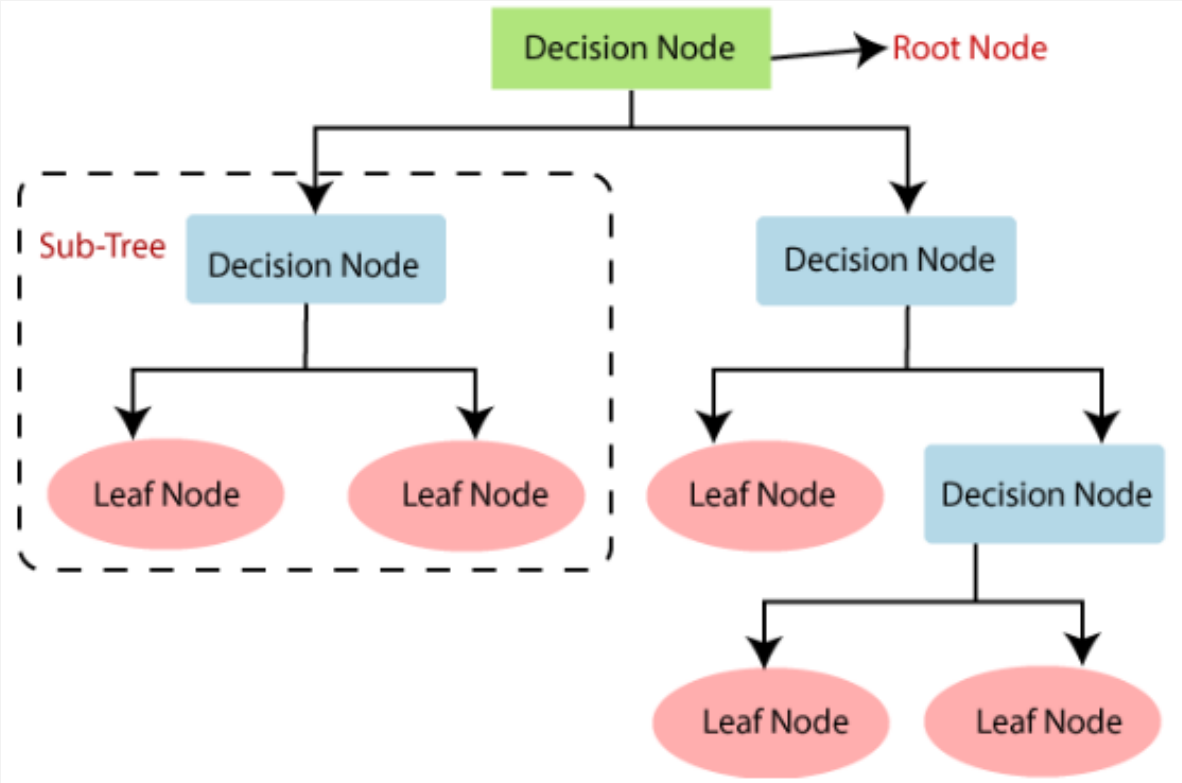
MACHINE LEARNING
NO GLYPH
UNIT II

DECISION TREES

INTRODUCTION

- Decision Trees is one of the Supervised Learning Technique.
- It can be used for both Classification and Regression problems
- Tree structure where internal nodes represent features of a dataset, branches represent decision rules, and each leaf node represents the outcome.
- In a decision tree, there are two nodes, which are the decision node and leaf node.
- Decision nodes are used to make any decisions and have multiple branches.
- Leaf nodes are the output of those decisions and do not contain any further branches.

General Structure of Decision Tree



TYPES OF DECISION TREES

- Binary Trees : Only two choices in each split. Can be non-uniform (uneven) in depth.
- N-way Trees or ternary trees : Three or more choices in at least one of its splits (3-way or 4-way).

APPROPRIATE PROBLEMS FOR DECISION LEARNING

- Instances are represented by attribute-value pairs.
- The target function has discrete output values.
- Disjunctive descriptions may be required.
- The training data may contain errors.
- The training data may contain missing attribute values.

ATTRIBUTE SELECTION MEASURE

Which attribute to be chosen at the root node of a decision tree ?

- This process is known as attribute selection measure. These measures are also used to split the root or child nodes. There are two such popular attribute selection measures :
- Information Gain
- Gini Index

INFORMATION GAIN

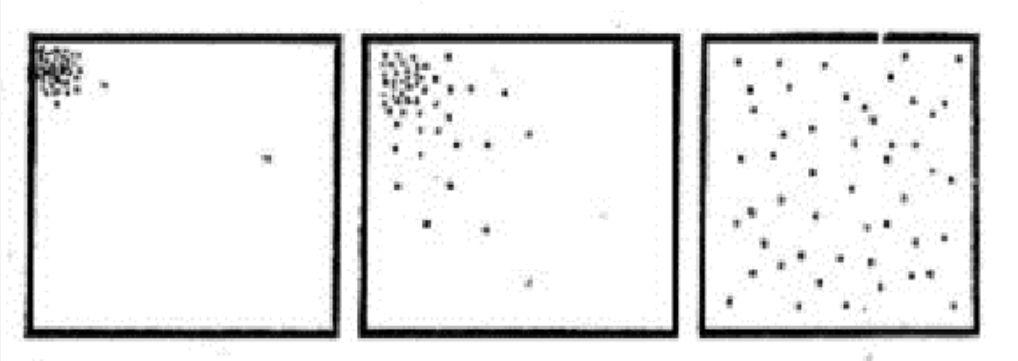
- Information Gain is the measurement of changes in entropy after segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula :
- Information Gain = Entropy(S) - [(Weighted Average) * Entropy(each feature)]

ENTROPY

- Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:
- Entropy(S)= -P(yes)log2 P(yes)- P(no) log2 P(no)
- Where S = Total number of samples, P(yes) = probability of yes and P(no) = probability of no

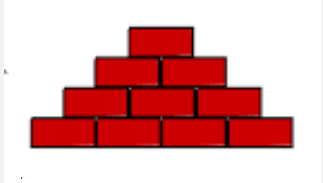
CONCEPT OF ENTROPY

CONCEPT OF ENTROPY

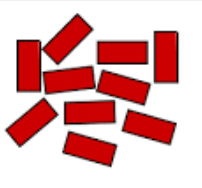


If a point represents a gas molecule, then which system has the more entropy?

How to measure? $\Delta S = \frac{\Delta Q}{T}$?

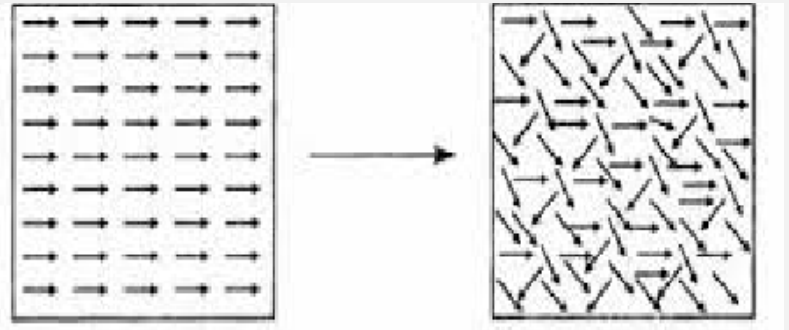


More **ordered**
ordered
less **entropy**
entropy



Less

higher



More organized or
organized or
ordered (less **probable**)
probable)

Less

disordered (**more**

CONCEPT OF ENTROPY

Universe!

What was its entropy value at its starting point?

AN OPEN CHALLENGE!

Roll No.	Assignment	Project	Mid-Sem	End-Sem
12BT3FP06	89	99	56	91
10IM30013	95	98	55	93
12CE31005	98	96	58	97
12EC35015	93	95	54	99
12GG2005	90	91	53	98
12MI33006	91	93	57	97
13AG36001	96	94	58	95
13EE10009	92	96	56	96
13MA20012	88	98	59	96
14CS30017	94	90	60	94
14ME10067	90	92	58	95
14MT10038	99	89	55	93

Roll No.	Assignment	Project	Mid-Sem	End-Sem
12BT3FP06	19	59	16	71
10IM30013	37	38	25	83
12CE31005	38	16	48	97
12EC35015	23	95	54	19
12GG2005	40	71	43	28
12MI33006	61	93	47	97
13AG36001	26	64	48	75
13EE10009	92	46	56	56
13MA20012	88	58	59	66
14CS30017	74	20	60	44
14ME10067	50	42	38	35
14MT10038	29	69	25	33

Two sheets showing the tabulation of marks obtained in a course are shown.

Which tabulation of marks shows the "good" performance of the class?

How you can measure the same?

GINI INDEX

- Gini Index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
- An attribute with the low Gini index should be preferred as compared to the high Gini index.
- It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
- Gini index can be calculated using the below formula:

  Gini Index= $1- \sum_j P_j^2$

What is the Gini index?

- The Gini index is a measure of impurity in a set of data. It is calculated by summing the squared probabilities of each class. A lower Gini index indicates a more pure set of data.

What is information gain?

- Information gain is a measure of how much information is gained by splitting a set of data on a particular feature. It is calculated by comparing the entropy of the original set of data to the entropy of the two child sets. A higher information gain indicates that the feature is more effective at splitting the data.

What is impurity?

Impurity is a measure of how mixed up the classes are in a set of data. A more impure set of data will have a higher Gini index.

How are Gini index and information gain related?

Gini index and information gain are both measures of impurity, but they are calculated differently. Gini index is calculated by summing the squared probabilities of each class, while information gain is calculated by comparing the entropy of the original set of data to the entropy of the two child sets.

When should you use Gini index and when should you use information gain?
Gini index and information gain can be used interchangeably, but there are some cases where one may be preferred over the other. Gini index is typically preferred when the classes are balanced, while information gain is typically preferred when the classes are imbalanced.

How do you calculate the Gini index for a decision tree?
The Gini index for a decision tree is calculated by summing the Gini indices of the child nodes. The Gini index of a child node is calculated by summing the squared probabilities of each class in the child node.

How do you calculate the information gain for a decision tree?
The information gain for a decision tree is calculated by comparing the entropy of the original set of data to the entropy of the two child sets. The entropy of a set of data is calculated by summing the probabilities of each class in the set multiplied by the log of the probability of each class.

ADVANTAGES AND LIMITATIONS OF INFORMATION GAIN AND GINI INDEX

The advantages of Gini index include:
- It is simple to calculate.
- It is interpretable.
- It is robust to overfitting.

The advantages of information gain include:
- It is more effective than Gini index when the classes are imbalanced.
- It is less sensitive to noise.

The disadvantages of Gini index include:
- It is not as effective as information gain when the classes are imbalanced.
- It can be sensitive to noise.

The disadvantages of information gain include:
- It is more complex to calculate.
- It is less interpretable.

DECISION TREE ALGORITHMS

The most notable and used decision tree algorithms are :

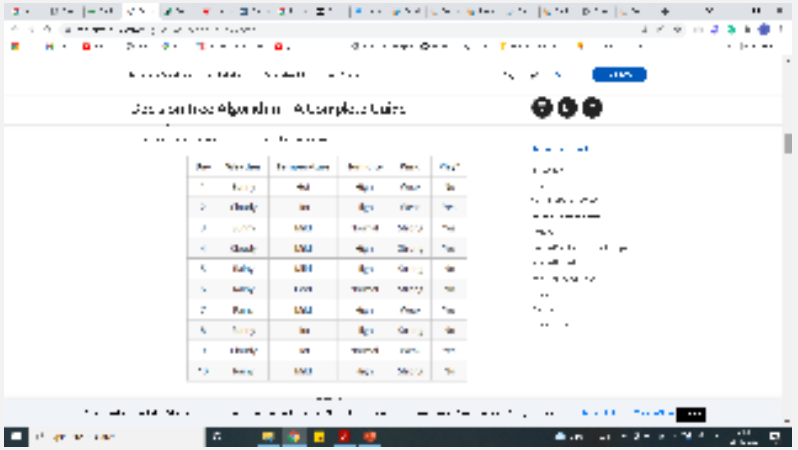
1. **Iterative Dichotomiser 3 (ID3)**
2. **C4.5**
3. **Classification and Regression Tree(CART)**

THE DECISION TREE ALGORITHM

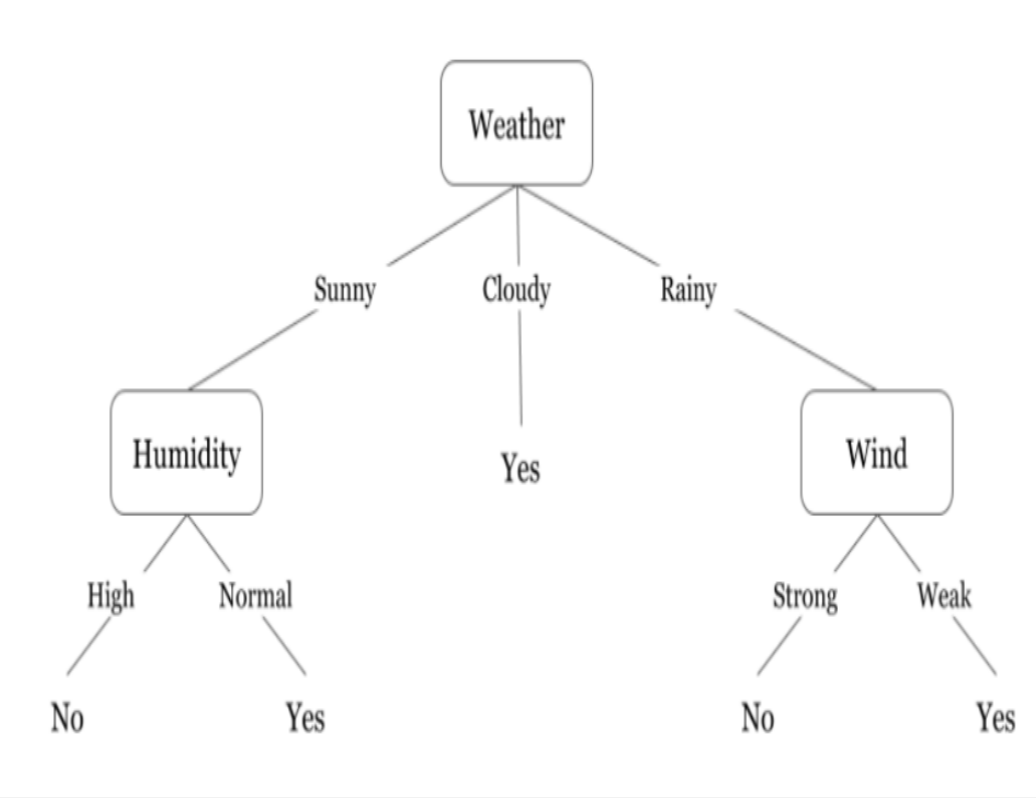
1. Begin the tree with the root node, says S, which contains the complete dataset.
2. Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**
3. Divide the S into subsets that contains possible values for the best attributes.
4. Generate the decision tree node, which contains the best attribute.
5. Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

EXAMPLE :

- The data has 6 attributes.
- Based on the given data, we want to find a set of rules to know what values of Weather, Temperature, Humidity and Wind determine whether or not to play outside.



- As shown in the given diagram, the tree will first ask what is the weather ? Then it moves on to the next feature which is wind and humidity.



TREE VALIDATION

- Confusion Matrix

		PREDICTED CLASS	
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

$$\text{Accuracy} = \frac{a+d}{a+b+c+d} = \frac{TP+TN}{TP+TN+FP+FN}$$

- Sometimes the cost of misclassification is not equal for both good and bad.
- We use a cost matrix along with confusion matrix.
- C(i | j) : Cost of misclassifying class j example as class i.

		PREDICTED CLASS	
ACTUAL CLASS	C(i|j)	**Class=Yes**	**Class=No**
	Class=Yes	C(Yes|Yes)	C(No|Yes)
	Class=No	C(Yes|No)	C(No|No)

DECISION TREE PRUNING

- *Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.*
- A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset.
- Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree pruning :
  - Pre-Pruning
  - Post-Pruning

ADVANTAGES OF DECISION TREES

- Decision Tree able to generate understandable rules
- They are able to handle both numerical and categorical attributes.
- They provide clear indication of which fields are most important for prediction or classification.

DISADVANTAGES OF DECISION TREES

- The process of growing a decision tree is computationally expensive. At each node, each candidate splitting field is examined before its best split can be found.
- Decision trees are less appropriate for estimation tasks where the goal is to predict the value of a continuous attribute.
- Decision trees are prone to errors in classification problems with many class and relatively small number of training examples.

SUPPORT VECTOR MACHINES (SVM)

SVM—SUPPORT VECTOR MACHINES

- A relatively new classification method for both linear and nonlinear data
- It uses a nonlinear mapping to transform the original training data into a higher dimension
- With the new dimension, it searches for the linear optimal separating **hyperplane** (i.e., "decision boundary ")
- With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane
- SVM finds this hyperplane using **support vectors** ("essential" training tuples) and **margins** (defined by the support vectors)

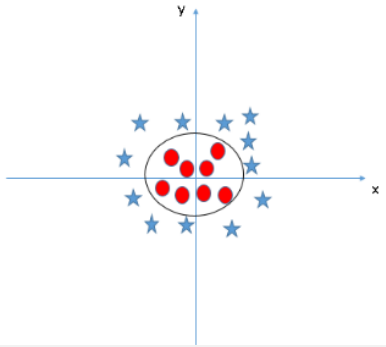
SVM

- "Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges.
- it is mostly used in classification problems.
- In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate.
- Then, we perform classification by finding the hyper-plane that differentiates the two classes very well (look at the below snapshot).

KERNEL TRICK

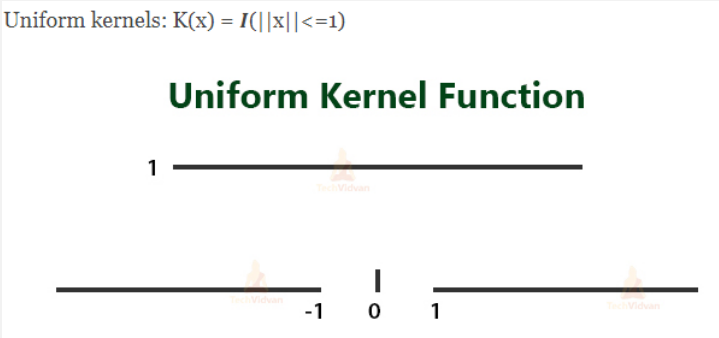
- The SVM kernel is a function that takes low dimensional input space and transforms it to a higher dimensional space i.e. it converts not separable problem to separable problem.
- it does some extremely complex data transformations, then finds out the process to separate the data based on the labels or outputs you've defined

When we look at the hyper-plane in original input space it looks like a circle:

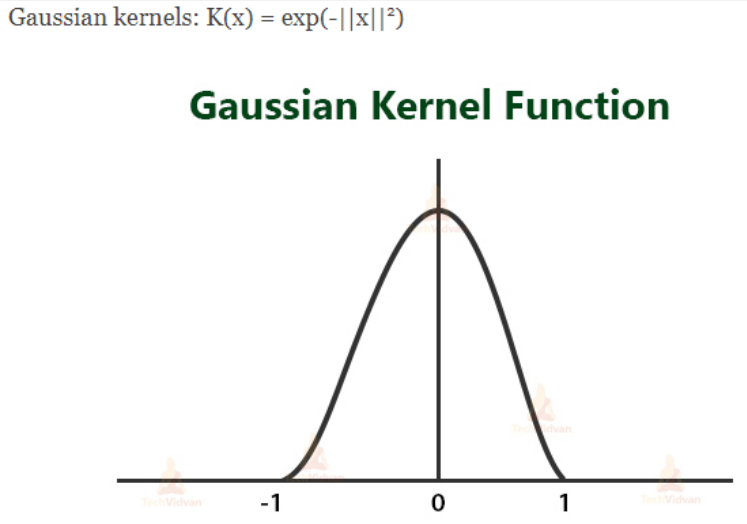


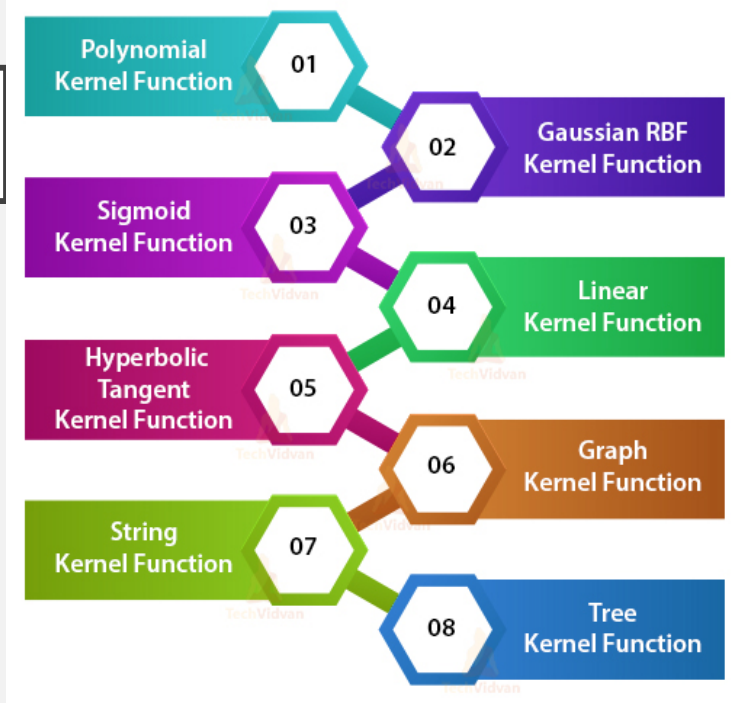
DIFFERENT KERNEL FUNCTIONS

Uniform kernels: K(x) = $I(||x||<=1)$



Gaussian kernels: K(x) = $\exp(-||x||^2)$



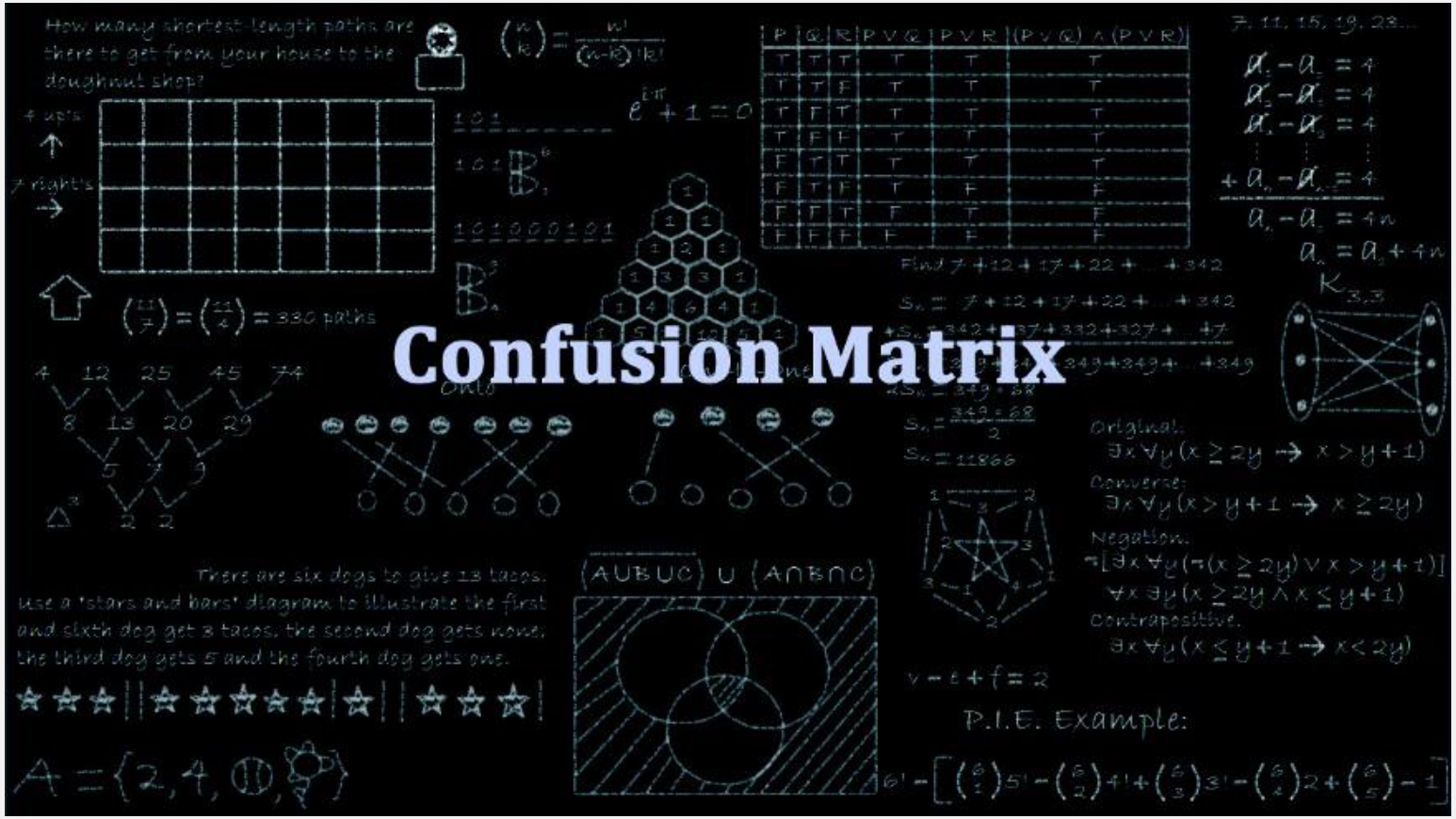


Linear	$K(x, y) = x \cdot y$
Sigmoid	$K(x, y) = \tanh(ax.y + b)$
Polynomial	$K(x, y) = (1 + x \cdot y)^d$
KMOD	$K(x, y) = a\left[\exp\left(\frac{\gamma}{\|\|x-y\|\|^2+\sigma^2}\right) - 1\right]$
RBF	$K(x, y) = \exp(-a\|\|x - y\|\|^2)$
Exponential RBF	$K(x, y) = \exp(-a\|\|x - y\|\|)$

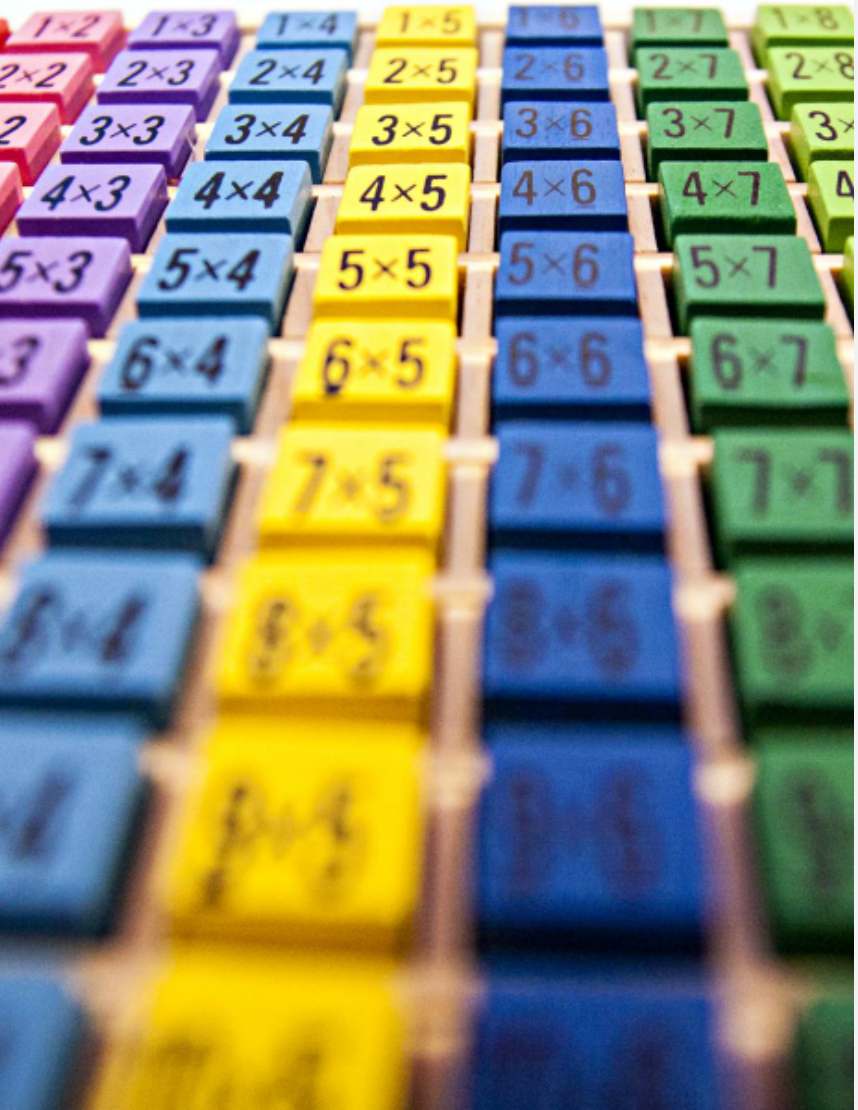
Pros and Cons associated with SVM

- **Pros:**
  - It works really well with a clear margin of separation
  - It is effective in high dimensional spaces.
  - It is effective in cases where the number of dimensions is greater than the number of samples.
  - It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

- **Cons:**
  - It doesn't perform well when we have large data set because the required training time is higher
  - It also doesn't perform very well, when the data set has more noise i.e. target classes are overlapping
  - SVM doesn't directly provide probability estimates, these are calculated using an expensive five-fold cross-validation. It is included in the related SVC method of Python scikit-learn library.

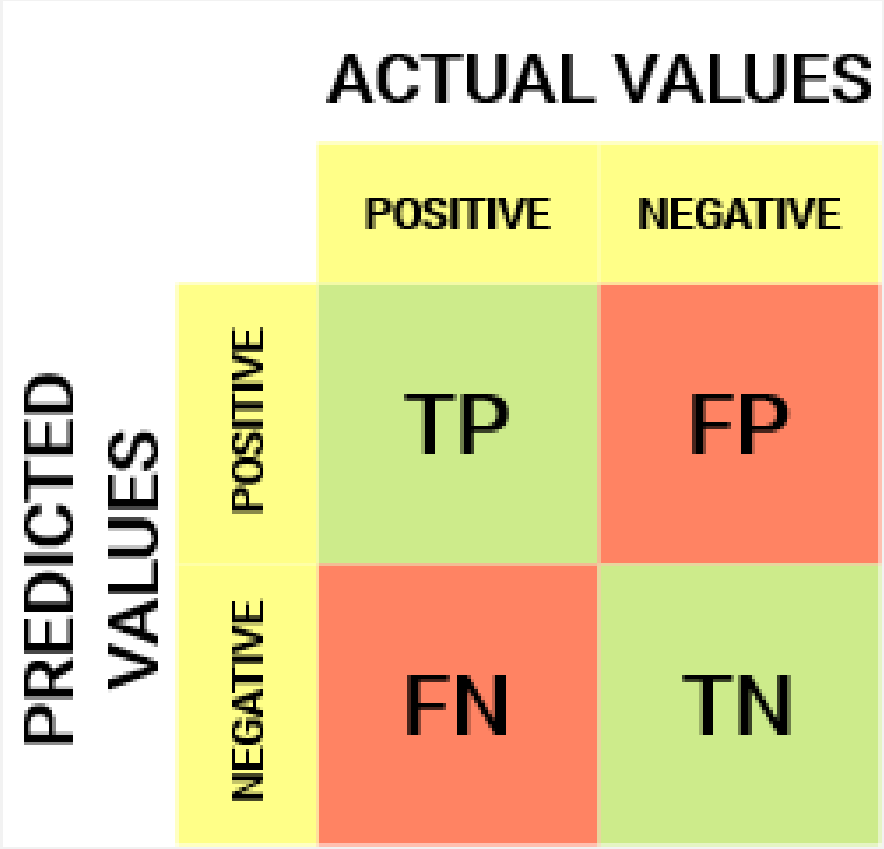


Confusion Matrix



THE CONFUSION ABOUT CONFUSION MATRIX?

"A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making."



CONFUSION MATRIX-EXAMPLE

Let's now define the most basic terms, which are whole numbers (not rates):

- **true positives (TP):** These are cases in which we predicted yes (they have the disease), and they do have the disease.
- **true negatives (TN):** We predicted no, and they don't have the disease.
- **false positives (FP):** We predicted yes, but they don't actually have the disease. (Also known as a "Type I error.")
- **false negatives (FN):** We predicted no, but they actually do have the disease. (Also known as a "Type II error.")

Actual class\Predicted class	C_1	$\neg C_1$
C_1	**True Positives (TP)**	**False Negatives (FN)**
$\neg C_1$	**False Positives (FP)**	**True Negatives (TN)**

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

CLASSIFICATION RATE/ACCURACY

- Accuracy or classification accuracy tells the number of correct predictions made by the model.
- It is the ratio of the number of correct predictions to the total number of input samples.
- However, there are problems with accuracy. It assumes equal costs for both kinds of errors. A 99% accuracy can be excellent, good, mediocre, poor, or terrible depending upon the problem.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

MISCLASSIFICATION RATE (ERROR)

- Accuracy or classification accuracy tells the number of wrong predictions made by the model.
- It is the ratio of the number of wrong predictions to the total number of input samples.

$$\text{Misclassification} = \frac{FP + FN}{TP + TN + FP + FN}$$

PRECISION / POSITIVE PREDICTIVE VALUE (PPV)

- When it predicts yes, how often is it correct?
- Precision is defined as the number of true positives divided by the number of true positives plus the number of false positives. Precision is about being precise.
- Out of all the predictive positive classes, how much we predicted correctly

$$\text{Precision} = \frac{TP}{TP + FP}$$

SENSITIVITY / RECALL / TRUE POSITIVE RATE

- When it's **actually yes, how often does it predict yes?**
- The recall is the fraction of positive events that were predicted correctly.
- 'When it is actually the positive result, how often does it predict correctly?'
- **High Recall** indicates the class is correctly recognized (a small number of FN).

$$\text{Recall} = \frac{TP}{TP + FN}$$

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

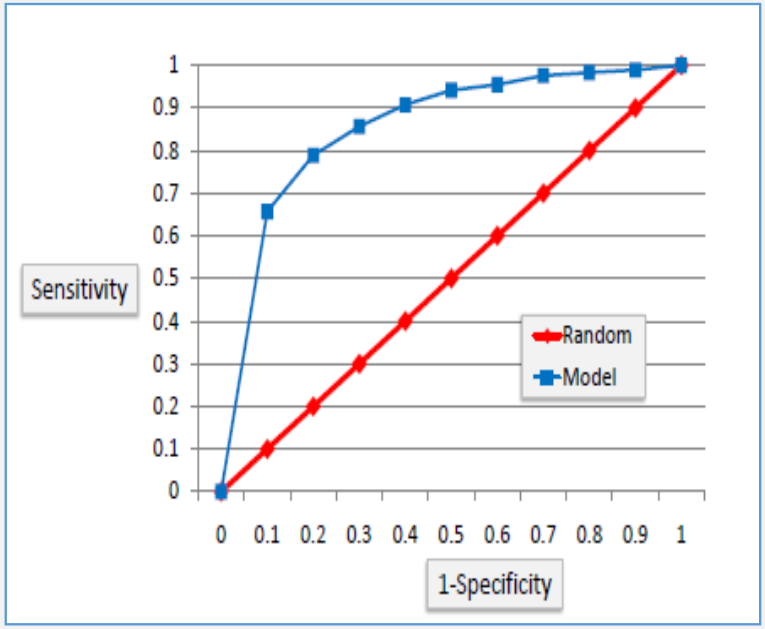
F1 SCORE

- The F1 score is the harmonic mean of the precision and recall, where
- F1 score reaches its best value at 1 (perfect precision and recall)
- F1 score reaches its worst value at 0
- A good F1 score means (Low FP & Low FN): Correctly identifying real threats, and not disturbed by false alarms
- F1 is usually more useful than accuracy, **especially for an uneven class distribution**
- Harmonic mean instead of arithmetic mean: HM punishes extreme values more

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

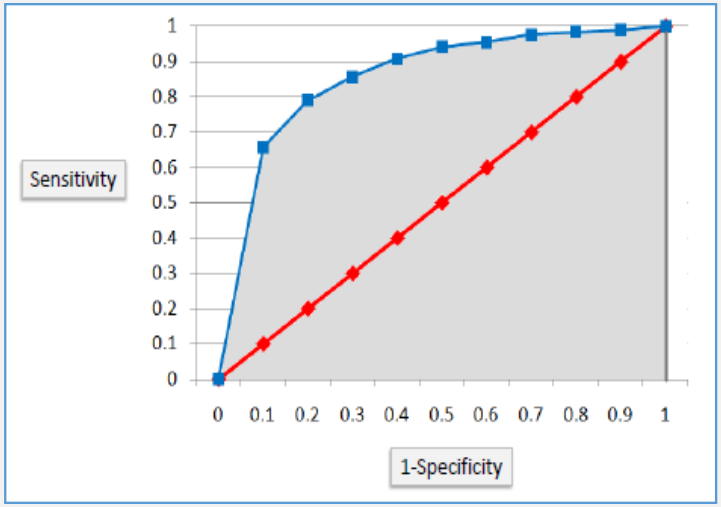
ROC CHART ..RECEIVER OPERATING CHARACTERISTIC

- The ROC chart: false positive rate (FPR) (1-specificity) on X-axis, the probability of target=1 when its true value is 0
- Against the true positive rate(TPR)(sensitivity) on Y-axis, the probability of target=1 when its true value is 1
- Ideally, the curve will climb quickly toward the top-left meaning the model correctly predicted the cases.
- The diagonal red line is for a random model



AREA UNDER THE CURVE (AUC)

- Area under ROC curve is often used as a measure of quality of the classification models.
- A random classifier has an area under the curve of 0.5, while AUC for a perfect classifier is equal to 1.
- In practice, most of the classification models have an AUC between 0.5 and 1



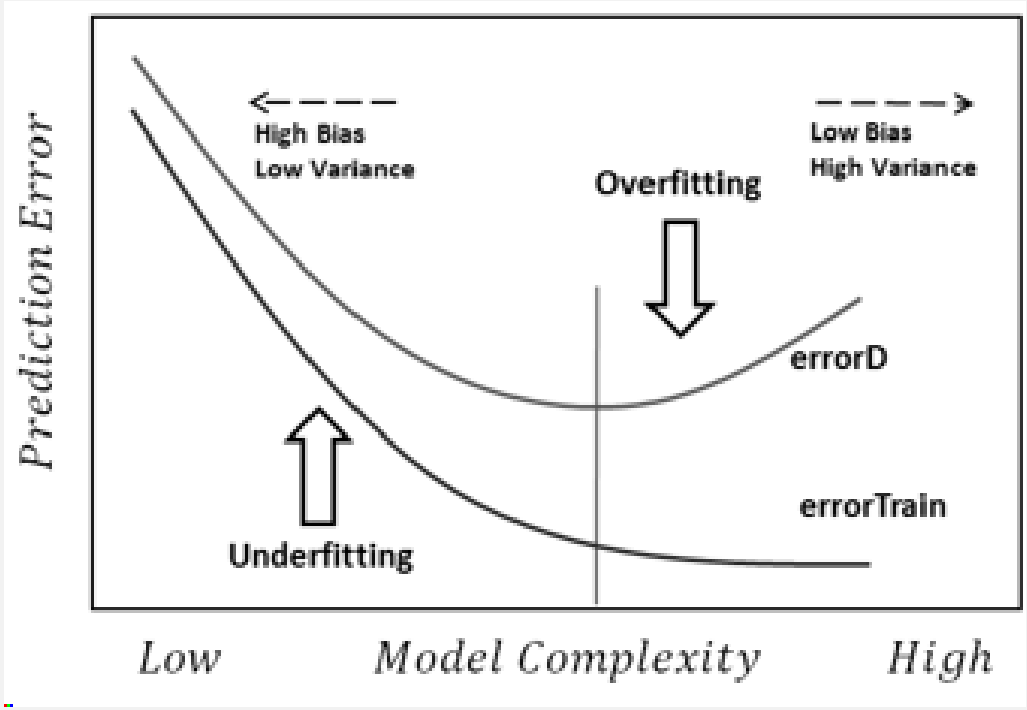
MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

ROC-AUC

- An area under the ROC curve of 0.8, for example, **means** that a randomly selected **case from the group with the target equals 1 has a score larger than** that for a **randomly chosen case from the group with the target equals 0** in 80% of the time.
- When a classifier cannot distinguish between the two groups, the area will be equal to 0.5 (the ROC curve will coincide with the diagonal).
- When there is a perfect separation of the two groups, i.e., no overlapping of the distributions, the area under the ROC curve reaches to 1 (the ROC curve will reach the upper left corner of the plot).

OVERFITTING

- Natural end of process in DT is 100% purity in each leaf
- This **overfits** the data, which end up fitting noise in the data
- Overfitting leads to low predictive accuracy of new data
- Past a certain point, the error rate for the validation data starts to increase

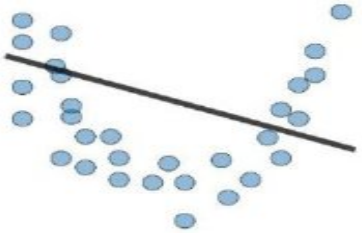
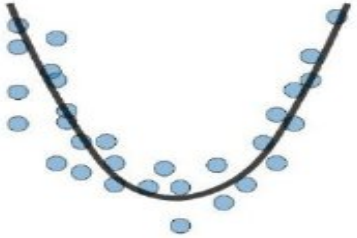
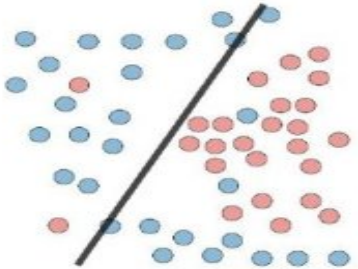
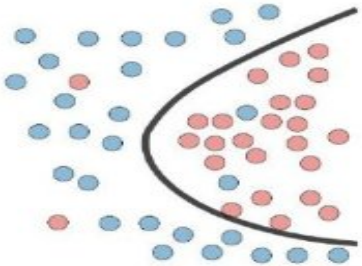
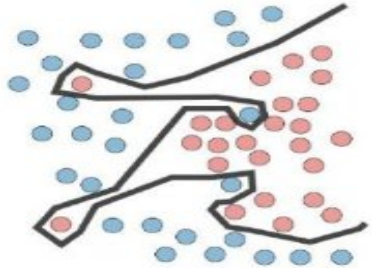
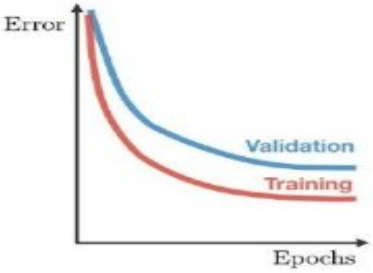
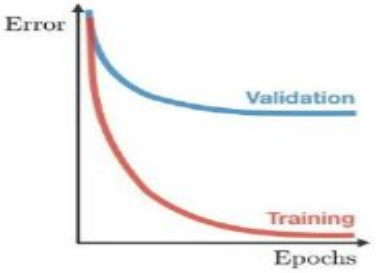


MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

OVERFITTING & UNDERFITTING

- Training a Model:
- **Overfit**: if the model is performing much better on train set but not performing well on cross-validation set
- **Overfit** is called as "**High Variance**" problem

- **Underfit**: if the model is not performing well on train set itself
- **Underfit** is considered as "**High Bias**" problem

	Underfitting	Just right	Overfitting
Symptoms	- High training error - Training error close to test error - High bias	- Training error slightly lower than test error	- Low training error - Training error much lower than test error - High variance
Regression			
Classification			
Deep learning			
Remedies	- Complexify model - Add more features - Train longer		- Regularize - Get more data

ENSEMBLING METHODS

BIAS VARIANCE

In machine learning, the *bias* is the difference between the prediction of the model and the ground truth value, while the *variance* is the error from sensitivity to small fluctuations in the training set.

Having high bias can cause an algorithm to miss the relevant relations between the target outputs and the features(underfitting) while high variance may result from an algorithm modeling the random noise in the training data (overfitting).

The *bias-variance tradeoff* is the property of a model that says that the bias in the estimated parameters can be reduced at the cost of increasing the variance of the parameter estimated across samples.

One way of resolving the bias-variance tradeoff is to use mixture models and ensemble learning. Generally, the main ensemble methods are *voting*, *stacking*, *bagging*, and *boosting*.

1. Ensemble methods

1.1. Voting

A *voting ensemble method* is a machine learning model that combines the predictions from different models to output a final prediction. For this ensemble method, the models should be different since it uses all the training data to train the models.

Figure 1. Training models of the voting ensemble method. **Ref:** Image by author.

For regression tasks, the output is the average of the predictions of the models. Instead, for classification tasks, there are two ways of estimating the final output: hard voting and soft voting. The former consists of taking the mode from all the predictions of the model (Figure 2). The latter uses the highest probability after averaging the probabilities for all the models (Figure 3).

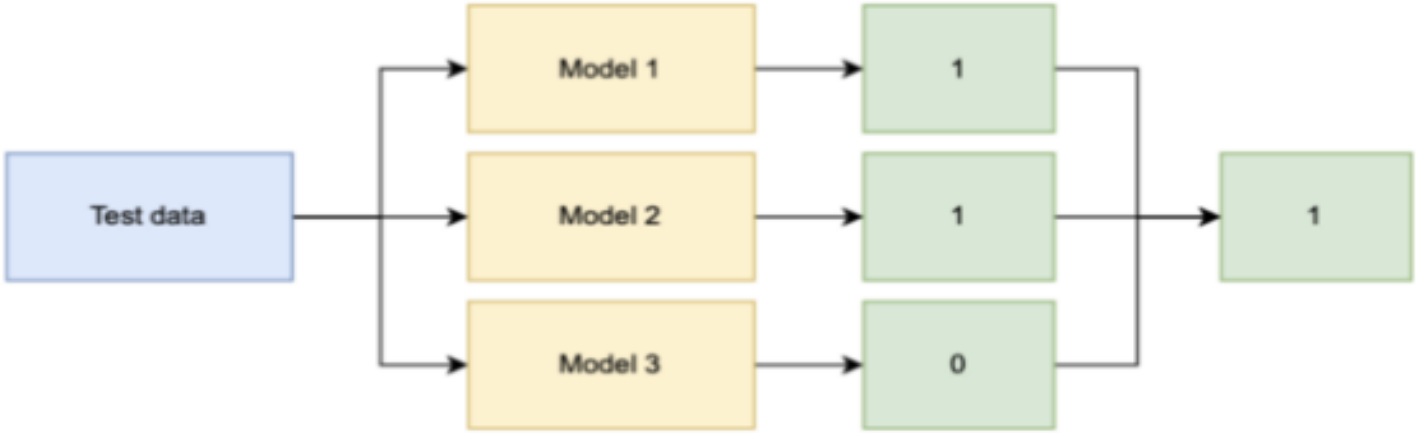


Figure 2. Hard voting. Ref: Image by author.

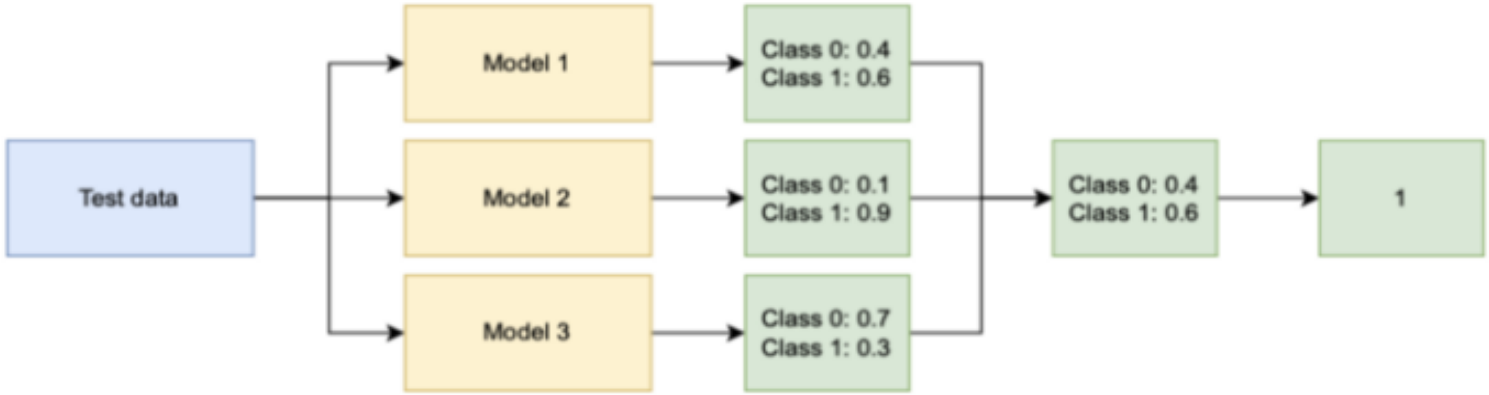


Figure 3. Soft voting. Ref: Image by author.

The main idea behind voting is to be able to generalize better by compensating for the errors of each model separately, especially when the models predict well on a predictive modeling task.

This ensemble method is particularly useful for stochastic machine learning models, such as the neural networks, as they result in a different model for each run. It is also useful when combining multiple fits of the same machine learning algorithm if it is observed that the same model performs well when using different hyperparameters.

1.2. Stacking

The stacking ensemble models are an extension of the voting ensemble models, as they use weighted voting of the contributing models to avoid all models contributing equally to the prediction.

The architecture of the stacking models involves *base-models* (models fitted on the training data) and a *meta-model* (model that learns how to combine the predictions of the base models). It is a common practice to use a linear regression model for regression tasks and a logistic regression model for classification tasks.

The meta-model is trained on the predictions made by base models on out-of-sample data. In other words, (1) data not used to train the base models are fed to the base models, (2) predictions are made from these base models, and (3) these predictions and the ground truth labels are provided to fit the meta-model.

The input of the meta-model depends on the task. For regression tasks, the input is the predicted value. For binary classification tasks, the input is typically the predicted value for the positive class. Lastly, for multiclass classification, it is usually the set of predicted values for all the classes.

There is a type of stacking model called *blending* commonly used in the literature. While stacking models are trained on out-of-fold predictions made during k-fold cross-validation, blending models are trained on predictions made on a holdout dataset.

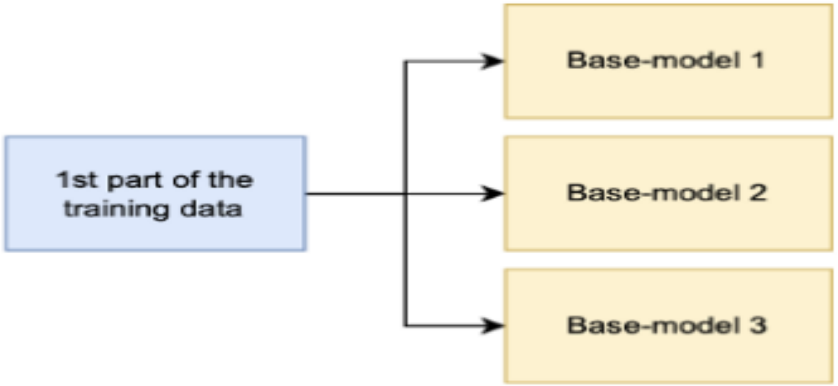


Figure 4. Training base-models of the stacking ensemble method. Ref: Image by author.

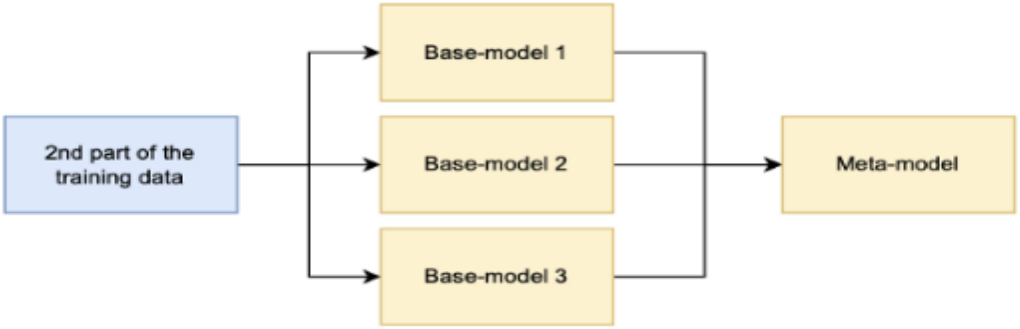


Figure 5. Training meta-models of the stacking ensemble method. Ref: Image by author.

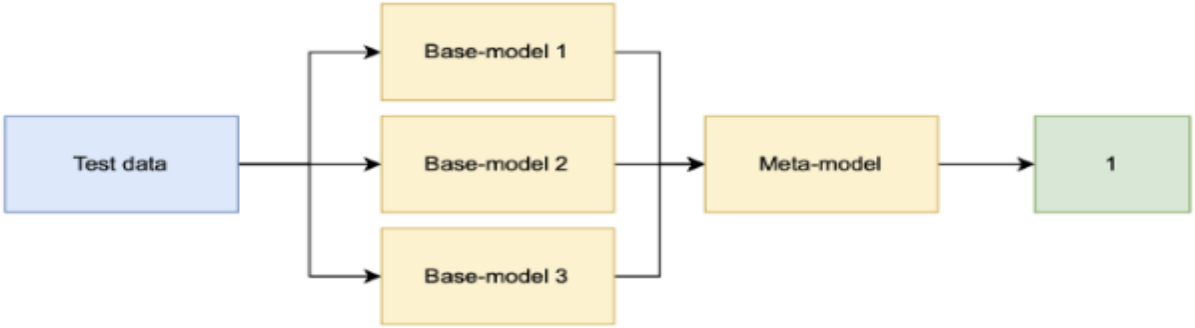


Figure 6. New data on stacking ensemble method. Ref: Image by author.

Stacking is designed to improve modeling performance, although is not guaranteed to result in an improvement in all cases. If any of the base models perform like the stacking ensemble method, this should be preferable use as it is easier to explain and maintain.

1.3. Bagging

Bagging is the process of sub-sampling training data to improve the generalization performance of a single type of classifier. This technique is effective on models which tend to overfit the dataset.

The method used to sub-sample the data is *bootstrapping* (the name *bagging* comes from *bootstrap* + *aggregating*). This method consists of doing random sampling with replacement over the data, which means that the subsets of the training data will overlap since we are not splitting the data but resampling it.

Once obtained the output of each of the models, the final prediction for each data can be obtained by doing *regression voting*, where predictions are the average of contributing models, or *classification voting*, where predictions are the majority vote of contributing models.

Some important notes to take into account when implementing this method are that (1) the hyperparameters of the classifier don't change from subsample to subsample, (2) the improvement is usually not really significant, (3) it is expensive as it may increase the computational costs by 5 or 10 times, and (4) this is a bias-reduction technique, so it does not help when you are variance-limited.

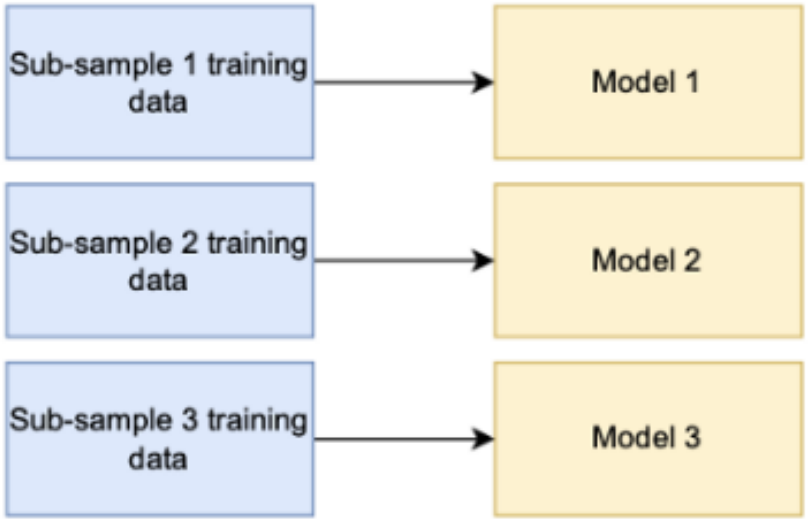


Figure 7. Training models of the bagging ensemble method. Ref: Image by author.

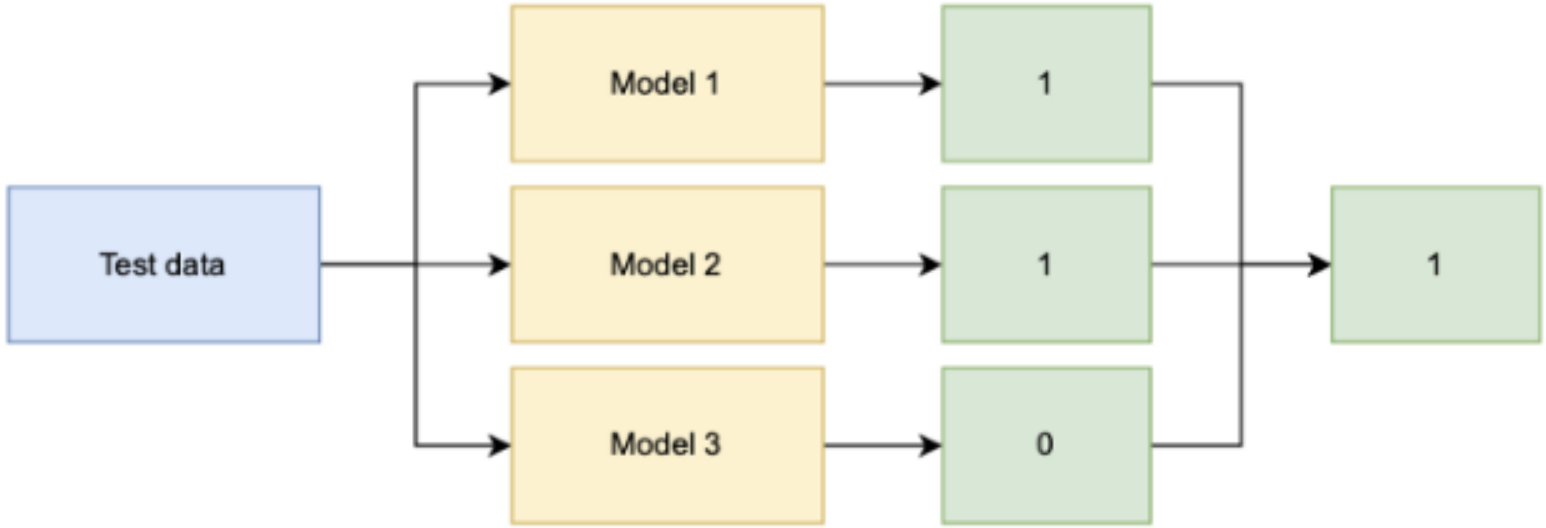


Figure 8. New data on bagging ensemble method. Ref: Image by author.

This ensemble method is especially useful when overfitting the data, but not when underfitting it, as it reduces the variance by generalizing better.

1.4. Boosting

Boosting is an ensemble method that builds the model by using weak learners in series. The main idea is that each successive model corrects the errors of its prior model.

Even though there are several types of boosting algorithms such as Gradient Boosting or XGBoosting, the first boosting algorithm developed for the purpose of binary classification was AdaBoost. The way AdaBoosting works is displayed in a simplified way in Figure 9, where the task is to classify the circles the squares based on the x and y features. The first model classifies the data points by generating a vertical separator line. But, as observed, it wrongly classifies some of the circles' data points. Hence, the second model focus on classifying these misclassified data points by increasing the weight of the wrongly classified data points. This process is iteratively done for the number of estimators defined when declaring the object. Since explaining the AdaBoost algorithm in more detail would take a whole article by itself, here I leave this video for anyone interested.

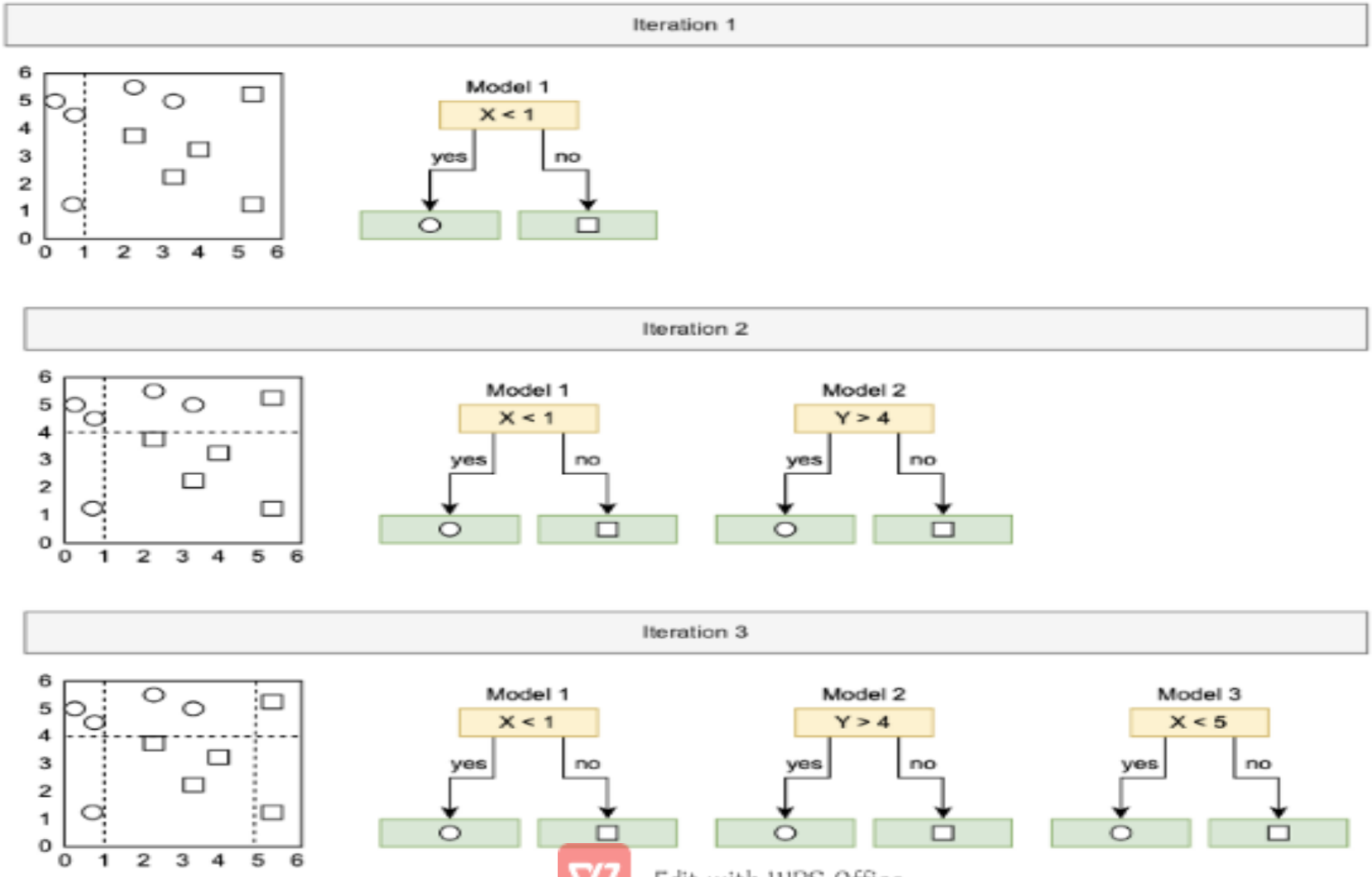
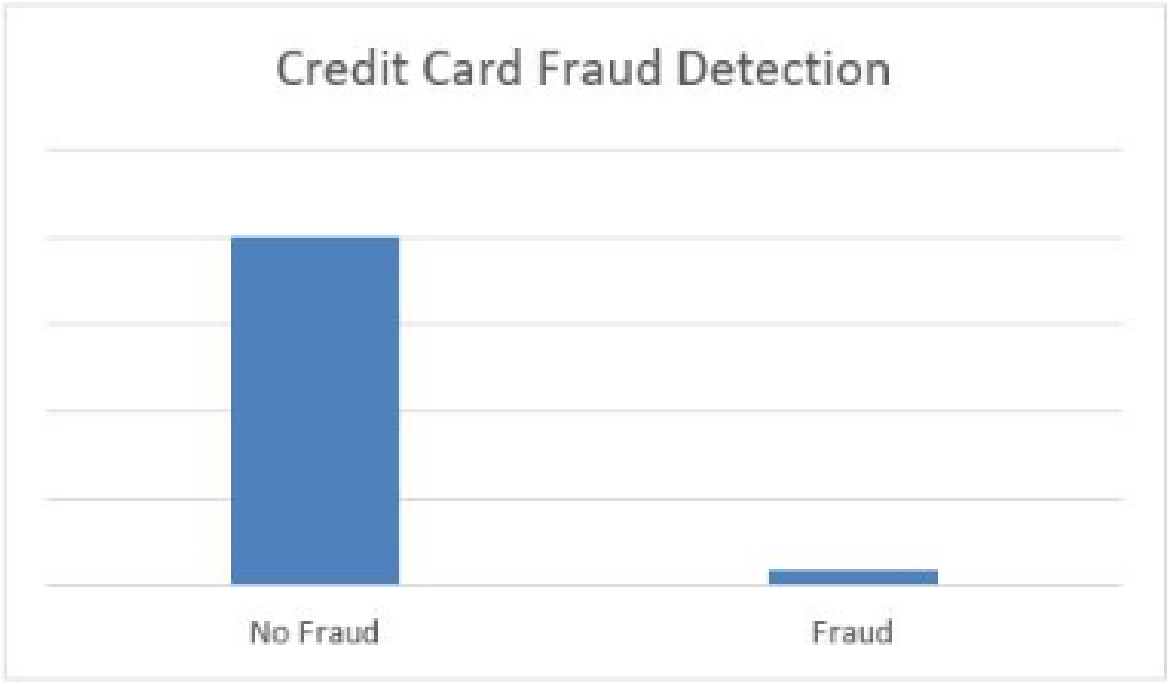


Figure 9. The training methodology for the boosting ensemble method. Ref: Image by author.

Boosting has been shown to achieve accurate results in the literature as they are a resilient method that curbs overfitting easily. However, it is sensitive to outliers as every classifier is obliged to fix the errors in the predecessors, and it is also difficult to scale up since each estimator bases its correctness on the previous predictors.

IMBALANCED DATA



The Problem with Imbalanced Classes

Most machine learning algorithms work best when the number of samples in each class are about equal. This is because most algorithms are designed to maximize accuracy and reduce error.

The Problem with Accuracy

Here we can use the DummyClassifier to always predict “not fraud” just to show how misleading accuracy can be.

1. Change the performance metric

As we saw above, accuracy is not the best metric to use when evaluating imbalanced datasets as it can be very misleading. Metrics that can provide better insight include:

- **Confusion Matrix:** a table showing correct predictions and types of incorrect predictions.
- **Precision:** the number of true positives divided by all positive predictions. Precision is also called Positive Predictive Value. It is a measure of a classifier's exactness. Low precision indicates a high number of false positives.
- **Recall:** the number of true positives divided by the number of positive values in the test data. Recall is also called Sensitivity or the True Positive Rate. It is a measure of a classifier's completeness. Low recall indicates a high number of false negatives.
- **F1: Score:** the weighted average of precision and recall.

2. Change the algorithm

While in every machine learning problem, it's a good rule of thumb to try a variety of algorithms, it can be especially beneficial with imbalanced datasets. Decision trees frequently perform well on imbalanced data. They work by learning a hierarchy of if/else questions and this can force both classes to be addressed.

3. Resampling Techniques — Oversample minority class

Our next method begins our resampling techniques.

Oversampling can be defined as adding more copies of the minority class. Oversampling can be a good choice when you don't have a ton of data to work with.

We will use the resampling module from Scikit-Learn to randomly replicate samples from the minority class.

4. Resampling techniques — Undersample majority class

Undersampling can be defined as removing some observations of the majority class. Undersampling can be a good choice when you have a ton of data -think millions of rows. But a drawback is that we are removing information that may be valuable. This could lead to underfitting and poor generalization to the test set.

We will again use the resampling module from Scikit-Learn to randomly remove samples from the majority class.

5. Generate synthetic samples

A technique similar to upsampling is to create synthetic samples. Here we will use imblearn's SMOTE or Synthetic Minority Oversampling Technique. SMOTE uses a nearest neighbors algorithm to generate new and synthetic data we can use for training our model.

Again, it's important to generate the new samples only in the training set to ensure our model generalizes well to unseen data.

Stuck on 80–85 % accuracy? There's a way to improve your model accuracy from a biased and imbalanced dataset by introducing Imblearn library.

Having an imbalanced dataset (imbalanced target variables) in a problem statement is always frustrating, and having a perfectly balanced dataset is always a myth.

Mostly in Medical Science/ Healthcare Machine Learning problems, the data set is mostly biased. In the case of the 'Breast Cancer Prediction', dataset, the data is biased towards the outcome 'Negative', as, according to the statistics, 92 out of 100,000 women were diagnosed with Breast Cancer in 2012. So, predicting outcome in such cases becomes very difficult as data becomes biased towards one particular class of outcome.

Since, it is a Healthcare problem our main goal is to reduce false Positive outcomes, as you cannot afford to let patients go away with a disease because of a biased algorithm. Since there are more 'Negatives' in a dataset, the Machine Learning Model becomes biased toward Negative Class. So, in some cases, it might predict 'Negative' for a 'Positive' Class.

There are many other cases where imbalanced datasets create the problem, I think in many World Problems Statements, this is true.

There are many ways to reduce the Bias Problem

1. **Improve Data Collection and Preprocessing Techniques:** Collect more data and give much more time to preprocessing by detecting outliers and segment the data according to balanced class.
2. **Up Sampling and Down Sampling:** If you have less data, then this technique is quite useful. Up(Over) Sampling is increasing the number of classes which is less in number by considering the data points closer to that of the original class. Down Sampling is the inverse of oversampling, i.e. reducing the number of classes having a higher number of data points.
3. **Use Specific Algorithm Properties:** This is helpful for some Machine Learning Algorithms where you can give weights to a particular class, which may decrease bias. For example, you have 70% -A Class and 30% -B Class, where you can give more weight on A class because Algorithm may tend to be more biased towards Class B.

KNN
K Nearest Neighbor Classification

Introduction

K-nearest neighbors (KNN) algorithm is a type of **supervised ML algorithm** which can be used for both classification as well as regression predictive problems.

❖ However, it is mainly used for classification predictive problems in industry.

There are three categories of learning algorithms:

1. **Lazy learning algorithm** – KNN is a lazy learning algorithm because it does not have a specialized training phase or model and uses all the data for training while classification.

2. **Non-parametric learning algorithm** – KNN is also a non-parametric learning algorithm because it doesn't assume anything about the underlying data.

3. **Eager learning algorithm** - Eager learners, when given a set of training tuples, will construct a generalization model before receiving new (e.g., test) tuples to classify.

KNN Algorithm:

K-nearest neighbors (KNN) algorithm uses 'feature similarity' to predict the values of new data points which further means that the new data point will be assigned a value based on how closely it matches the points in the training set. We can understand its working with the help of following steps –

Step 1 – For implementing any algorithm, we need dataset. So during the first step of KNN, we must **load the training as well as test data**.

Step 2 – Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.

Step 3 – For each point in the test data do the following –

3.1 – Calculate the distance between test data and each row of training data with the help of any of the method namely: **Euclidean, Manhattan or Hamming distance**. The most commonly used method to calculate distance is Euclidean.

3.2 – Now, based on the distance value, sort them in ascending order.

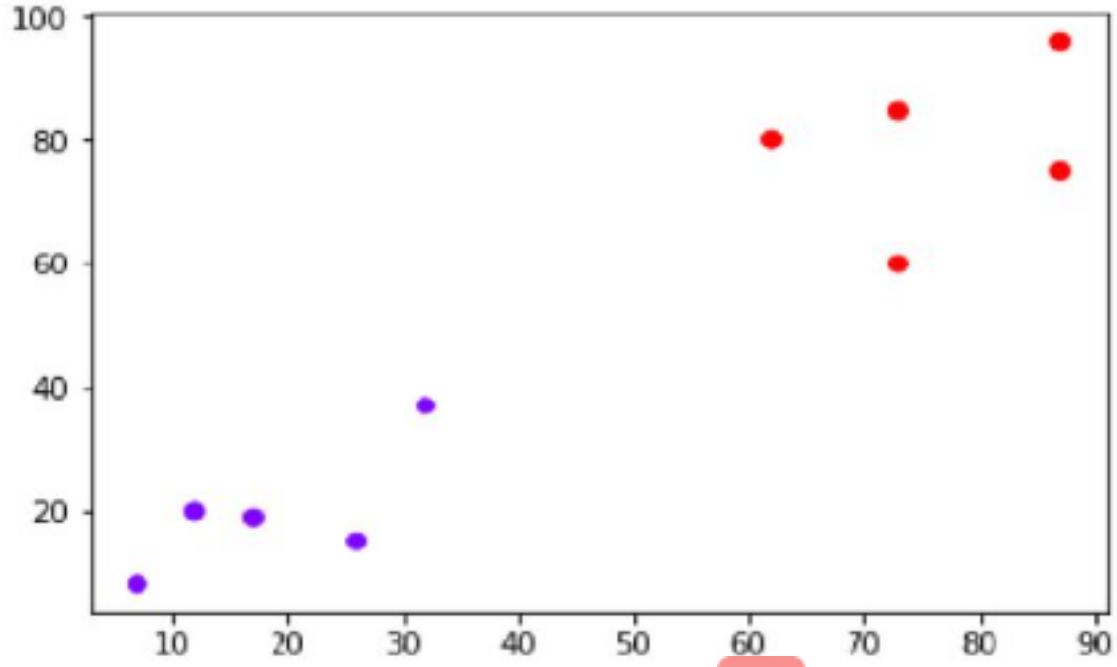
3.3 – Next, it will choose the top K rows from the sorted array.

3.4 – Now, it will assign a class to the test point based on most frequent class of these row

Step 4 – End

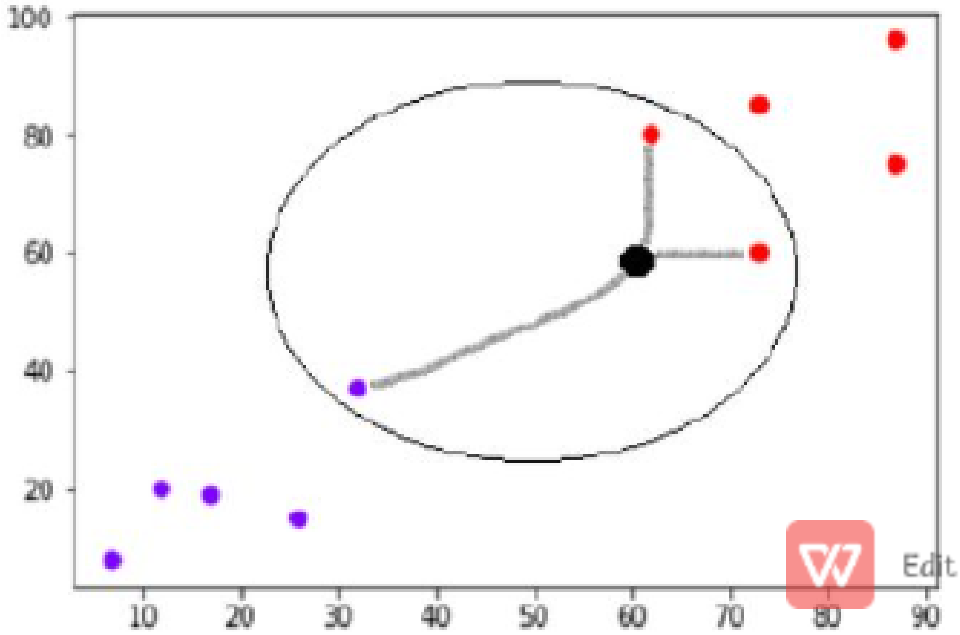
Example-1:

- The following is an example to understand the concept of K and working of KNN algorithm
- Suppose we have a dataset which can be plotted as follows:



Example-1 (Conti..)

- Now, we need to classify new data point with black dot (at point 60,60) into blue or red class. We are assuming K = 3 i.e. it would find three nearest data points. It is shown in the following diagram:



- We can see in the beside diagram the three nearest neighbors of the data point with black dot. Among those three, two of them lies in Red class hence the **black dot** will also be assigned in red class.

Advantages

1. **No Training Period:** KNN is called Lazy Learner (Instance based learning). It does not learn anything in the training period. It does not derive any discriminative function from the training data. It stores the training dataset and learns from it only at the time of making real time predictions. This makes the KNN algorithm much faster than other algorithms that require training e.g. Linear Regression etc.

2. Since the KNN algorithm requires **no training** before making predictions, new data can be added seamlessly which will **not impact the accuracy of the algorithm**.

3. **KNN is very easy to implement**. There are only two parameters required to implement KNN i.e. the value of K and the distance function (e.g. Euclidean or Manhattan etc.)

Dis advantages

1. **Does not work well with large dataset:** In large datasets, the cost of calculating the distance between the new point and each existing points is huge which degrades the performance of the algorithm.
2. **Does not work well with high dimensions:** The KNN algorithm doesn't work well with high dimensional data because with large number of dimensions, it becomes difficult for the algorithm to calculate the distance in each dimension.
3. **Need feature scaling:** We need to do feature scaling (standardization and normalization) before applying KNN algorithm to any dataset. If we don't do so, KNN may generate wrong predictions.
4. **Sensitive to noisy data, missing values and outliers**: KNN is sensitive to noise in the dataset. We need to manually impute missing values and remove outliers.

Conclusion:

KNN is an effective **machine learning algorithm** that can be used in credit scoring, prediction of cancer cells, image recognition, and many other applications. The main importance of using **KNN** is that it's easy to implement and works well with small datasets.