Unit 1- Regression with Tidymodels

Nana Boateng

Study Objectives

After completing this unit, students should be able to:

- Apply packages within the tidymodels ecosystem
- Perform linear regression with tidymodels
- Perform logistic regression with tidymodels
- Fit a model using the parsnip package
- Perform model evaluation with the yardstick package

Overview of Predictive & Prescriptive Analytics

- Big Data Analytics (BDA) facilitates the analyses of complex data and the generation of insights which hitherto were impossible (Mehta and Pandit, 2018) using traditional data processing software.

- Types of BDA:
  - Descriptive analytics
  - Diagnostic Analytics
  - Predictive Analytics
  - Prescriptive Analytics

Types of BDA

Machine Learning Methods

- Supervised Methods
- Unsupervised Methods

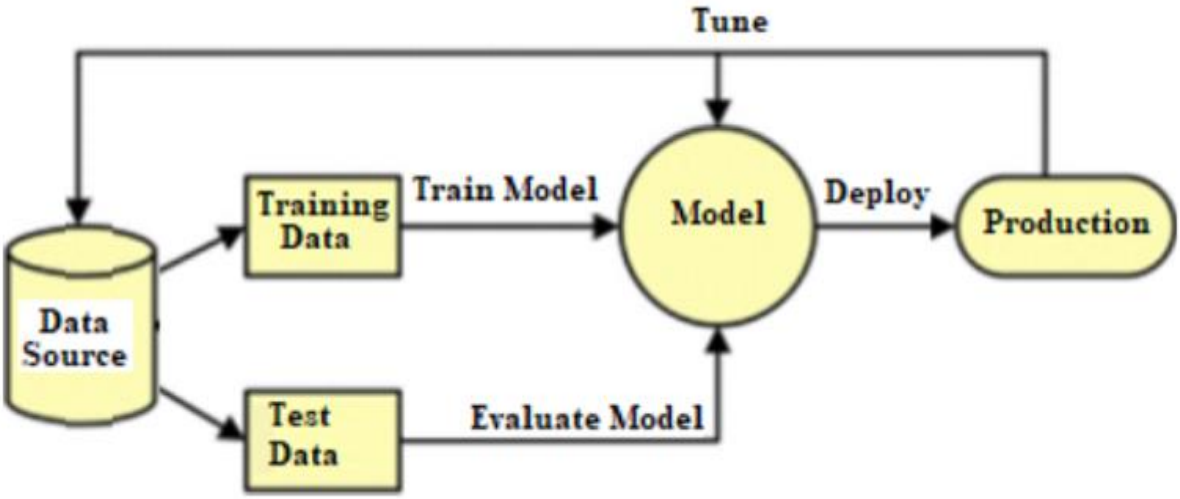
Unsupervised Machine Learning

- Unsupervised machine learning is a type of machine learning algorithm that works without labeled data. Instead, it uses algorithms that group data into clusters according to similar characteristics without any prior knowledge or labels. Examples of unsupervised machine learning algorithms include k-means clustering, hierarchical clustering, and affinity propagation.
- These algorithms are often used for exploratory data analysis to find meaningful patterns and trends in data, as well as for anomaly detection. They can also be used to generate new features from existing data.

Supervised Machine Learning

- Supervised machine learning is a type of machine learning algorithm that uses a known dataset (labeled data) to train a model to make predictions. It is based on the concept of learning from example data, where the model is given an input and an associated output value, which it learns from and uses to infer meaning from the data, then predict an output value for new data.
- Examples of supervised machine learning algorithms include k-nearest neighbors (KNN), decision trees, logistic regression, and support vector machines (SVMs), etc.

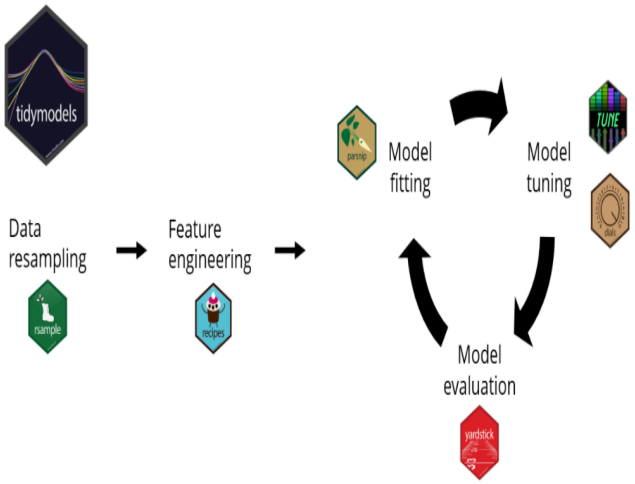
Supervised Machine Learning Workflow



Machine Learning Packages for Modelling -Tidymodels

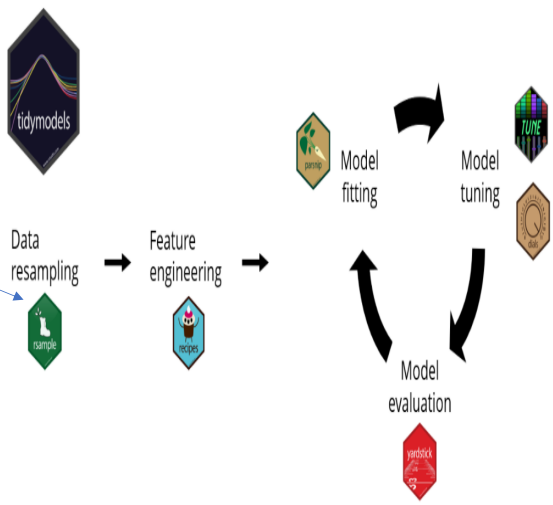
- Tidymodels is a collection of R packages designed to support machine learning model development.

- Tidymodels is designed to easily iterate over model fitting, tuning, and evaluation, all with a unified R syntax!



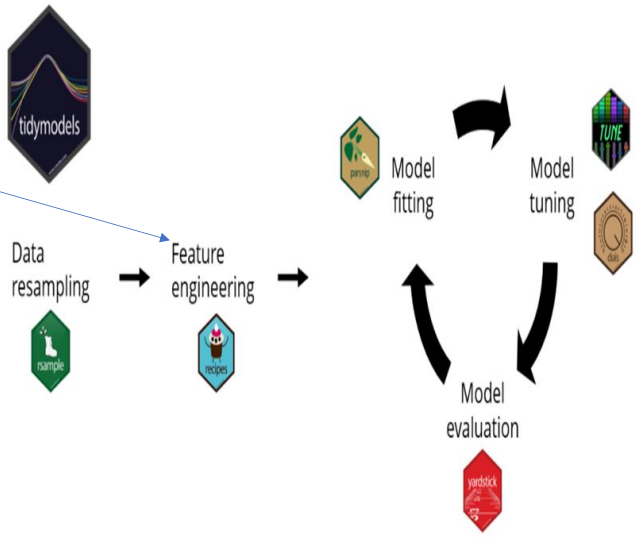
Modelling with Tidymodels Package

- **Rsample Package**
  - The rsample package supports data resampling, and is used for creating random subsets of a dataset for different activities in the modelling process.



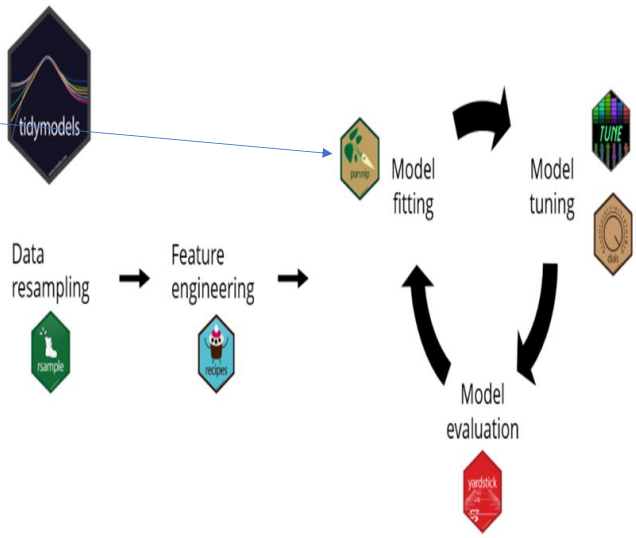
Modelling with Tidymodels Package

- **Recipes Package**
  - The *recipes package* contains functions for transforming data for modeling. This step is often called feature engineering.



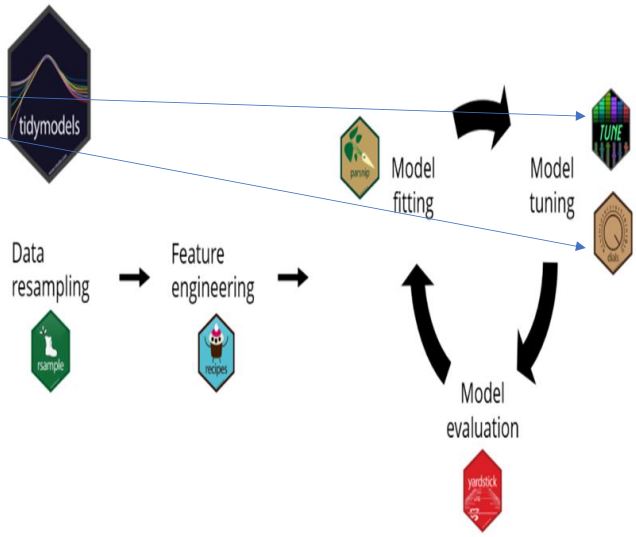
Modelling with Tidymodels Package

- **Parsnip Package**
  - The *parsnip package* is an interface to the vast modeling libraries available in R. It is used for specifying and fitting models as well as obtaining model predictions.



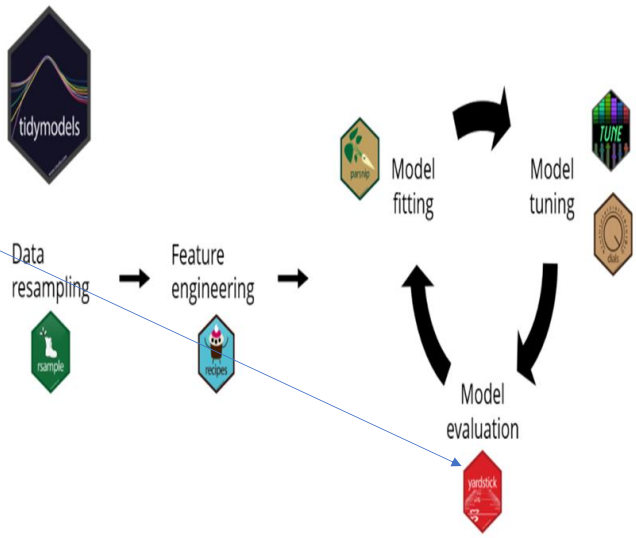
Modelling with Tidymodels Package

- **Tune & Dials Package**
  - The *tune and dials packages* provide functionality for fine-tuning models in order to achieve optimal prediction accuracy.



Modelling with Tidymodels Package

- **Yardstick Package**
  - The yardstick package provides metrics for evaluating the quality of model predictions.



Supervised Machine Learning & Tidymodels

- Tidymodels is primarily used for supervised machine learning, where algorithms learn patterns from labeled data.
- There are two types of supervised machine learning:
  - **Regression Models** (predicting a quantitative outcome e.g. selling price of a home)
  - **Classification Models** (predicting a categorical outcome e.g. whether an employee will leave the company)

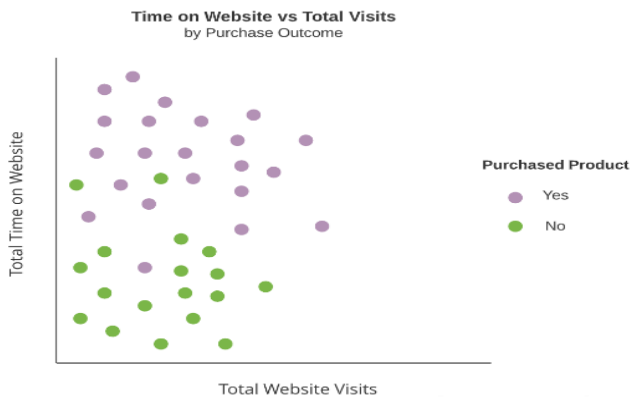
left_company	miles_from_home	salary
no	1	84500
yes	10	64820
no	5	76490
yes	19	68540

- *left_company* is an outcome variable
- *miles_from_home* and *salary* are predictor variables

Classification Models with Tidymodels

- Classification models predict categorical outcome variables, for example, predicting whether a customer will make a purchase or not based on the number of times they spend on a website.

purchased	total_time	total_visits
yes	800	3
yes	978	7
no	220	4
no	124	5
yes	641	4



Linear Regression Model

For instance, predicting highway fuel efficiency (hwy) using city fuel efficiency (cty) as predictor.

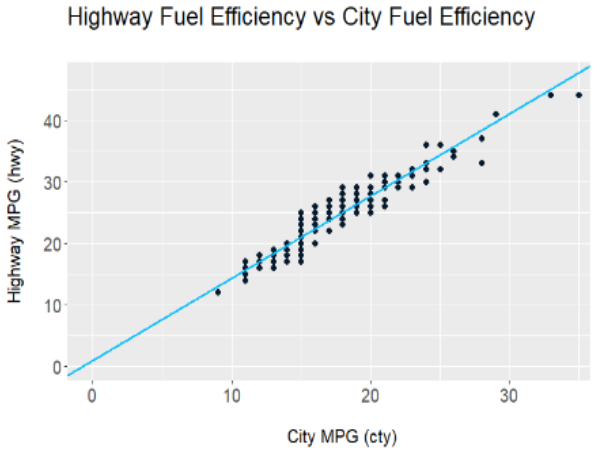
$$hwy = \beta_0 + \beta_1 cty$$

Model parameters

- β_0 is the intercept
- β_1 is the slope

Estimated paramters from training data

$$hwy = 0.77 + 1.35(cty)$$



Linear Regression Formula

Model formulas in `parsnip`

- Used to assign column roles
  - Outcome variable
  - Predictor variables

General form

```
outcome ~ predictor_1 + predictor_2 + ...
```

Shorthand notation

```
outcome ~ .
```

Predicting `hwy` using `cty` as a predictor variable

```
hwy ~ cty
```

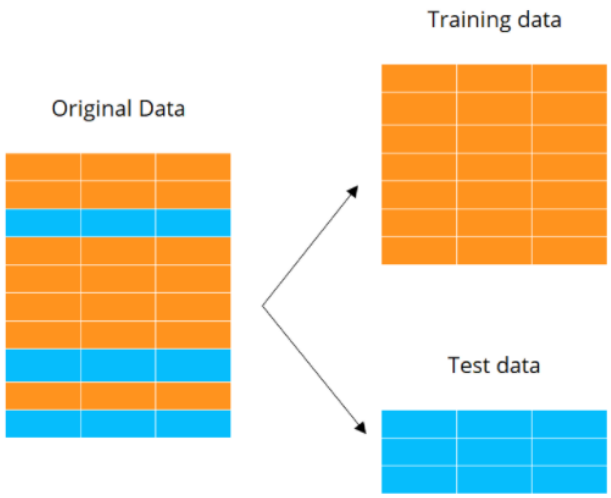
Data Resampling [Data Partitioning into Training & Testing Data]

Training Data

- Feature Engineering
- Modelling

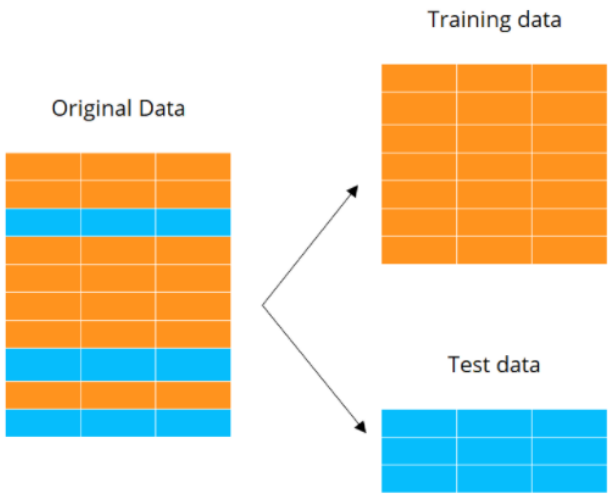
Test Data

- Estimate model performance on new data



Data Resampling

- The first step in modelling is to randomly split the original data into training and test datasets. This guards against a phenomenon known as **overfitting**, where a model memorizes the patterns in a dataset and then performs poorly on new data.
- Commonly 75% of the data is allocated into training and 25% into test.



Data Resampling

To begin the modelling process, import your data and load the tidymodels package.

- initial-split ()
  - Specifies instructions for creating training and test datasets
- prop argument
  - Specifies the proportion to place into training
- strata argument
  - Provides stratification by the outcome variable
- Pass the split object into the training model
- Pass the split object into the test model

```
> #Data Partioning
> home_split <- initial_split(home_sales, prop = 0.7,
+                             strata = selling_price)
> #Populating the training dataset
> home_training <- home_split%>%
+   training()
> #Populating the testing dataset
> home_test <- home_split%>%
+   testing()
```


Linear Regression with Tidymodels

- **Parsnip package** - used for fitting models and calculating predictions.

1. Specify model type
   - Linear Regression or any other model type
2. Specify the engine
   - Different engine correspond to different underlying packages
3. Specify the mode
   - Either Regression or Classification

Model Fitting – Linear Regression

- To fit a linear regression model, we start by defining a parsnip model object named lm_model. We use the linear_reg() function to create a linear regression model object.
- Then we pass it into the set_engine function where we specify the 'lm' engine.
- Finally we pass the results into the set_mode() function where we specify 'regression' since we are predicting a numeric outcome variable.
- To train our model, we pass lm_model to the fit() function and provide a model formula and the data on which to train the model.

```
lm_object <- linear_reg()%>%
  set_engine('lm')%>%
  set_mode('regression')
```

```
lm_fit <- lm_object%>%
  fit(selling_price ~ home_age + sqft_living,
      data = home_training)
```


Estimating Model Parameters

- The tidy() function is used to obtain model parameters.

```
tidy(lm_fit)
```

```
A tibble: 3 x 5
term         estimate std.error statistic   p.value
<chr>           <dbl>     <dbl>     <dbl>     <dbl>
(Intercept)  291685.    7617.       38.3  8.46e-201
home_age      -1631.     179.       -9.09 4.87e- 19
sqft_living     104.       2.75     37.7  1.45e-196
```

Making Predictions

- Model predictions are obtained by passing the trained model, lm_fit, to the predict() function.
- The ***new_data*** argument specifies the dataset on which to predict new values and is typically set to the testing dataset.
- The predict() function returns standardized output that is always a tibble, has the same row order as the data in the new_data argument, and a column named dot-pred with model predictions.

```
home_prediction <- lm_fit%>%
  predict(new_data = home_test)

home_prediction
```

```
> home_prediction
# A tibble: 450 x 1
     .pred
     <dbl>
 1 538984.
 2 409108.
 3 473008.
 4 531720.
 5 508887.
 6 425268.
 7 500288.
 8 486203.
 9 522379.
10 399248.
# ... with 440 more rows
```


Adding Predictions to the Test Data

- To evaluate model performance, we will need to add the model predictions to the test dataset.
  - The ***bind_cols() function*** can be used to combine multiple data frames along the column axis.

```
home_test_results <- home_test %>%
  select(selling_price, home_age, sqft_living) %>%
  bind_cols(home_prediction)
```

```
> home_test_results
# A tibble: 450 x 4
   selling_price home_age sqft_living   .pred
           <dbl>    <dbl>       <dbl>   <dbl>
 1        487000       10        2540 538984.
 2        355000       19        1430 409108.
 3        425000       11        1920 473008.
 4        535000        3        2360 531720.
 5        495000        3        2140 508887.
 6        381000       25        1680 425268.
 7        525000       28        2450 500288.
 8        540000       22        2220 486203.
 9        533500       17        2490 522379.
10        355000       26        1445 399248.
# ... with 440 more rows
```

Evaluating Model Performance -Yardstick Package

Root Mean Standard Error (RMSE)

- A common performance metric for regression models is the root mean squared error, or RMSE.

Root Mean Square Error (RMSE)

$$RMSE = \sqrt{\frac{1}{n}\sum (actual - predicted)^2}$$

- The RMSE estimates the average prediction error of a model and is calculated with the **rmse() function.**

```
home_test_results%>%
  rmse(truth = selling_price, estimate = .pred)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard      47638.
```


Evaluating Model Performance -Yardstick Package

R Square () Function

- Another important regression metric is R squared, also known as the coefficient of determination.
- R squared measures the squared correlation between actual and predicted values and ranges from 0 to 1, where 1 indicates that all predictions equal the true outcome values. R squared is calculated with the rsq() function and takes the same arguments as the rmse() function.

```
home_test_results%>%
  rsq(truth = selling_price, estimate = .pred)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rsq     standard       0.655
```

Streamlining Model Fitting -Last Fit()Function

- The last_fit() function is used to streamline the model fitting and evaluation process in tidymodels. It takes a parnsip model object, model formula, and rsample data split object and performs the following steps.
  - It creates training and test datasets
  - fits the model to the training data
  - calculates metrics and predictions on the test data
  - returns an object with all the results.

Streamlining Model Fitting -Last Fit()Function

- To fit our linear regression model on the mpg data, we pass the lm_model parsnip object to last_fit(), specify our model formula, and provide the data split object.

```
lm_last_fit <- lm_object%>%
  last_fit(selling_price ~ home_age + sqft_living,
           split = home_split)
```

Streamlining Model Fitting -Collecting Metrics

- Once the model is trained with the last_fit() function, we pass the lm_last_fit object to the collect_metrics() function to get a tibble with calculated metrics on the test data.
- The default metrics for regression models are RMSE and R squared and are always stored in the column named dot-estimate.
- We get the same performance metrics on the home_test data as before, just with a lot less work!

```
lm_last_fit%>%
  collect_metrics()

A tibble: 2 x 4
.metric .estimator .estimate .config
<chr>   <chr>          <dbl> <chr>
rmse    standard   47638.    Preprocessor1_Model1
rsq     standard       0.655 Preprocessor1_Model1
```


Streamlining Model Fitting -Collecting Predictions

- To collect the model predictions on the test data, we pass the lm_last_fit object to the collect_predictions() function and obtain a tibble with the test dataset predictions.
- The predictions column is always named dot-pred. The outcome variable, House_Price in our case, is also included along with other row identifier columns.

```
lm_last_fit%>%
  collect_predictions()

# A tibble: 450 x 5
   id               .pred  .row selling_price .config
   <chr>            <dbl> <int>         <dbl> <chr>
 1 train/test split 538984.     1        487000 Preprocessor1_Model1
 2 train/test split 409108.     7        355000 Preprocessor1_Model1
 3 train/test split 473008.     9        425000 Preprocessor1_Model1
 4 train/test split 531720.    10        535000 Preprocessor1_Model1
 5 train/test split 508887.    12        495000 Preprocessor1_Model1
 6 train/test split 425268.    18        381000 Preprocessor1_Model1
 7 train/test split 500288.    19        525000 Preprocessor1_Model1
 8 train/test split 486203.    20        540000 Preprocessor1_Model1
 9 train/test split 522379.    34        533500 Preprocessor1_Model1
10 train/test split 399248.    37        355000 Preprocessor1_Model1
# ... with 440 more rows
```

LOGISTIC REGRESSION

Logistic Regression

- Logistic Regression is used for classifying categorical variables. It is a popular classification algorithm which creates a linear separation between outcome categories. The logistical function takes the value of zero and one which can be interpretated as probability (Husseina, Hamdanb and Zugharc, 2020).
- The estimated probability of a logistic regression is determined as:

$$p = \frac{e^{\beta 0+\beta 1x1}}{1 + e^{\beta 0+\beta 1x1}}$$

- The assumption of linearity, normality, and homoscedasticity that generally apply to regressions do not apply to logistic regression.

Use Cases of Logistic Regression

Credit Scoring

- *Logistic regression can be used to assign a credit score to loan applicants based on relevant financial data.*

Predicting Customer Churn

- *Companies can use logistic regression to predict which customers are likely to defect from their loyalty programs.*

Detecting Fraud

- *Using logistic regression, organizations can identify the probability that a particular transaction is fraudulent.*

Use Cases of Logistic Regression

Diagnosing Medical Conditions

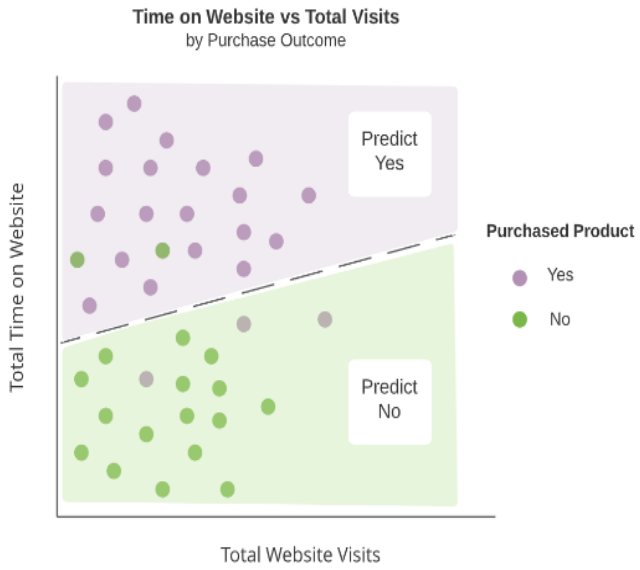
- Logistic regression can be used to analyze medical data and diagnose conditions.

Predicting Mortgage Risk

- Financial institutions use logistic regression to assess the risk associated with granting mortgages.

Logistic Regression in R -Using Tidymodels Package

- **Goal**: Create distinct, non-overlapping regions along a set of predictor variable values.
  - Predict the same categorical outcome in each region



Logistic Regression

- It is a popular classification algorithm which creates a linear separation between outcome categories

Logistic Regression -Model Specification

- Model specification in Parnsnip

- Logistic_reg()
- The function provides the general interface to logistic regression model in parsnip

Model Specification:

```
Logistic_Model<- logistic_reg()%>%
        set_engine ('glm')%>%
        set_mode('classification')
```

Logistic Regression -Model Fitting

- Once the model is specified, the fit() function is used for training
- Pass model object to fit()
- Specify Model Formula

```
logistic_fit <- Logistic_Model%>%
  fit(Attrition ~ . ,
      data = hrt_training)
```


Logistic Regression -Predicting Outcome Categories

- Uses the *predict ()* function
- *new_data* specifies the dataset on which to predict new values
- *type*
  - *'class'* provides categorical predictions
  - *'prob'* provides probability estimates

```
class_preds <- logistic_fit%>%
  predict(new_data = hrt_testing,
          type = 'class') #class provides categorical predictions

class_preds

#Estimated Probabilities
#set the type argument to prob

prob_preds <- logistic_fit%>%
  predict(new_data = hrt_testing,
          type = 'prob')
```

Assessing Model Fit

- Binary Classification
  - Outcome with two levels

Positive Class
Event of interest to predict
- Yes

Negative Class
Event of interest to predict

- No

```
> #In tidymodels, outcome variable needs to be a factor
> #First level is positive class
> levels(hrt_results[['Attrition']])
[1] "Yes" "No"
```


Assessing Model Fit - Confusion Matrix

- A confusion matrix is a matrix with counts of all combinations of actual and predicted outcome values.

TP = True Positive

FP = False Positive

FN = False Negative

TN = True Negative

		Truth	
		Positive (+)	Negative (-)
Predicted	Positive (+)	TP	FP
	Negative (-)	FN	TN

Assessing Model Fit - Confusion Matrix

Correct Predictions:

- True Positives (TP)
- True Negatives (TN)

Classification Errors

- False Positives (FP)
- False Negatives (FN)

Predicted	Truth: Positive (+)	Truth: Negative (-)
Positive (+)	TP	FP
Negative (-)	FN	TN

```
> conf_mat(hrt_results,
+          truth= Attrition,
+          estimate = .pred_class)
          Truth
Prediction Yes  No
       Yes  14   4
       No   46 305
```


Assessing Model Fit - Confusion Matrix

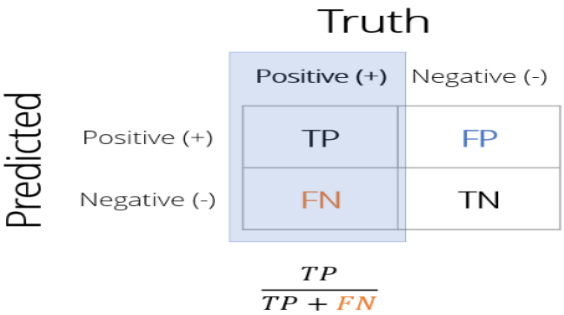
- Accuracy = $\frac{TP+TN}{TP+TN+FP+FN}$

```
> accuracy(hrt_results,
+      truth= Attrition,
+      estimate = .pred_class)
# A tibble: 1 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.864
```

Assessing Model Fit - Confusion Matrix

Sensitivity

Sensitivity is the proportion of all positive cases that were correctly classified.



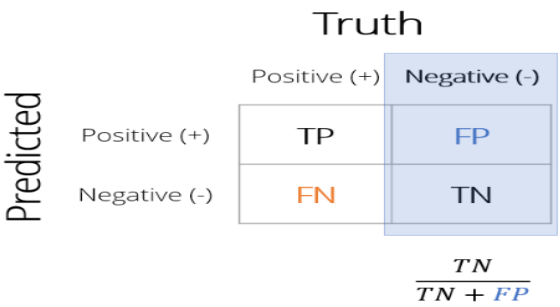
$$\frac{TP}{TP + FN}$$

```
> sens(hrt_results,
+          truth= Attrition,
+          estimate = .pred_class)
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 sens    binary         0.233
```


Assessing Model Fit - Confusion Matrix

Specificity

Specificity is the proportion of all negative cases that were correctly classified.



$$\frac{TN}{TN + FP}$$

```
> spec(hrt_results,
+          truth= Attrition,
+          estimate = .pred_class)
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 spec    binary         0.987
```

Visualizing Model Performance

Probability Thresholds

- Default probability threshold in binary classification is 0.5
- If the estimated probability of the positive class is greater than or equal to 0.5, the positive class is predicted.

```
> hrt_results
    Attrition .pred_class   .pred_Yes  .pred_No
1          No          No 0.071737942 0.9282621
2          No         Yes 0.524622795 0.4753772
3          No          No 0.087557964 0.9124420
4          No          No 0.070000701 0.9299993
5          No          No 0.291641361 0.7083586
6          No          No 0.009870775 0.9901292
7          No          No 0.177125254 0.8228747
8          No          No 0.211727707 0.7882723
9          No          No 0.495056725 0.5049433
10         No          No 0.102047215 0.8979528
11         No          No 0.063226719 0.9367733
12        Yes          No 0.366814326 0.6331857
13         No          No 0.057289035 0.9427110
14        Yes         Yes 0.539802424 0.4601976
```


Visualizing Model Performance -Exploring Performance Across Thresholds

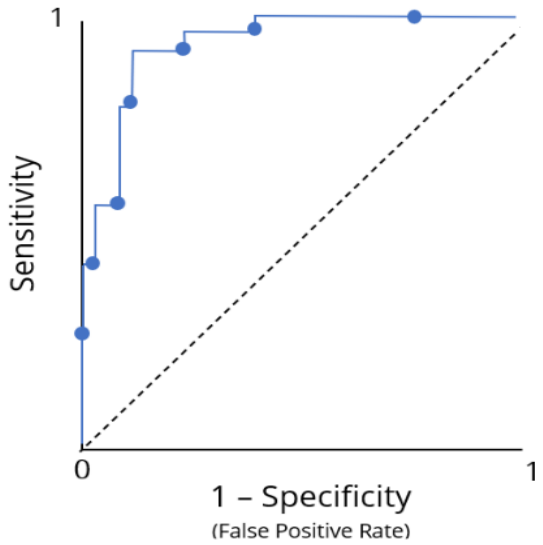
- How does a classification Model perform across a range of thresholds?
- A way to do it is to use the probability thresholds in the .pred_yes column of the test dataset results.
- Then, calculate the specificity and sensitivity for each

threshold	specificity	sensitivity
0	0	1
0.11	0.01	0.98
0.15	0.05	0.97
...	...	...
0.84	0.89	0.08
0.87	0.94	0.02
0.91	0.99	0
1	1	0

Visualizing Model Performance - Receiver Operating Characteristic Curve (ROC)

- The ROC Curve is used to visualize performance across probability thresholds.

- Sensitivity vs. (1-Specificity) across unique thresholds in test set results
- Shows proportion correct among actual positives vs. proportion incorrect among actual negatives.



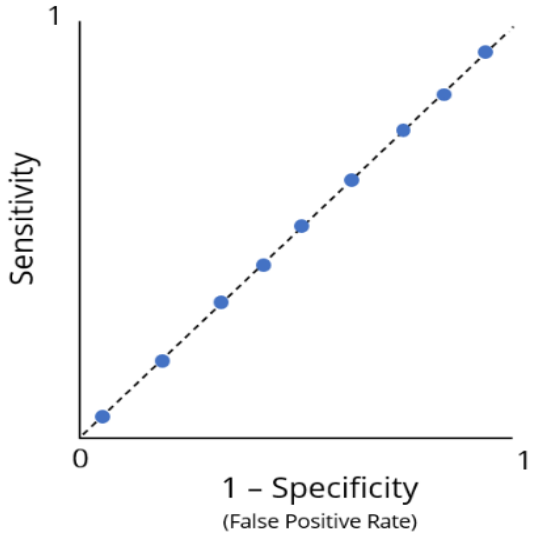
Visualizing Model Performance - Receiver Operating Characteristic Curve (ROC)

Optimal performance is at the point (0, 1)

- Ideally, a classification model produces points close to left upper edge across all thresholds

Poor performance

- Sensitivity and (1 - specificity) are equal across all thresholds
  - Corresponds to a classification model that predicts outcomes based on the result of randomly flipping a fair coin



Summarising the ROC Curve

Area Under Curve (AUC)

- The Area Under Curve (AUC) captures the ROC curve information of a classification model in a single number.

```
> #Calculate the ROC Area Under Curve(AUC)
> hrt_results%>%
+   roc_auc(truth = Attrition, .pred_Yes)
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 roc_auc binary         0.779
```


Final Thoughts…

- Data is split into Training & Testing Data for Machine Learning Purposes
- Supervised Machine Learning models comprises Classification & Regression Models.
- Classification Models are used for predicting categorical outcomes
- Regression Models are used for predicting numerical outcomes
- Logistic Regression models are classification models
- ROC curve gives a diagrammatic display of model performance
- Area Under Curve (AUC) is a number (between 0 and 1) used in determining the effectiveness of the classification model

Unit 2- Feature Engineering

Nana Boateng

Study Objectives

After completing this unit, students should be able to:

i. Explain the pre-processing steps for model fitting
ii. Transform numeric variable by dealing with multicollinearity
iii. Create dummy variables for nominal predictor variables
iv. Perform a complete modelling workflow

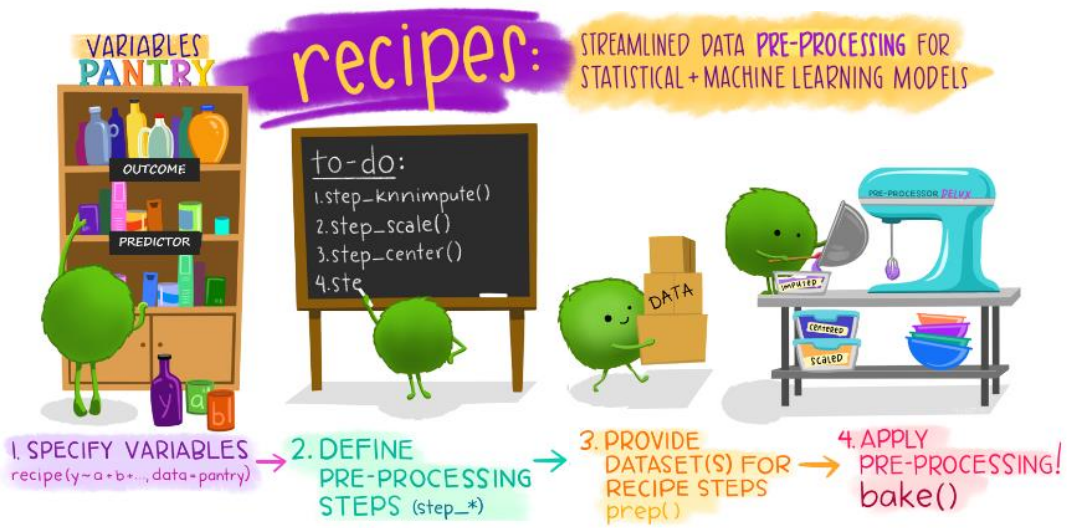
Feature Engineering

- Feature engineering is the process of transforming data to a format that is suitable for machine learning algorithms.

Feature Engineering (with Recipes Package)

- Feature engineering is accomplished with the recipes package. It is designed to help with all stages of feature engineering, which include:
  - assigning variable roles to the columns of our data.
  - defining preprocessing tasks and data transformations.
  - training our data transformations.
  - applying them to new data sources.

Feature Engineering (with Recipes Package)



Feature Engineering (with Recipes Package) - Specifying Variable Types & Roles

Define column roles

- Assign outcome or predictor role to all variables

Determine variable data types

- Numeric data
- Categorical data

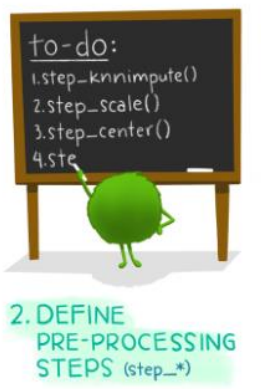
Accomplished with the `recipe()` function

Feature Engineering (with Recipes Package) - Data Pre-processing Steps

Add required data preprocessing steps

- Imputation of missing data
- Data transformations
  - Centering and scaling numeric variables
- Creating new variables
  - Calculating ratios of variables
- And many more...

Each step is added with a unique `step_*()` function



Feature Engineering (with Recipes Package) - Training Pre-processing Steps

`recipe` objects are trained on a data source, typically the training dataset

- Data transformations are estimated
  - Mean and standard deviation of numeric columns for centering and scaling
  - Formulas for creating new columns are stored for applying to new data

Recipes are trained with the `prep()` function

Feature Engineering (with Recipes Package) - Applying Recipes to New Data

Apply all trained data preprocessing transformations

- To the training and test datasets for modeling
- To new sources of data for future predictions
  - Machine learning algorithms require the same data format as was used during training to predict new values

Recipes are applied with the `bake()` function

Transforming Nominal Predictors

Nominal data must be transformed to numeric data for modeling

One-Hot Encoding

- Maps categorical values to a sequence of [0/1] indicator variables
- Indicator variable for each unique value in original data

department
finance
marketing
technology

→

department_finance	department_marketing	department_technology
1	0	0
0	1	0
0	0	1

Transforming Nominal Predictors

Dummy Variable Encoding

- Excludes *one value* from original set of data values
  - ***n*** distinct values produce (***n - 1***) indicator variables
- Preferred method for modeling
  - Default in `recipes` package

department
finance
marketing
technology

→

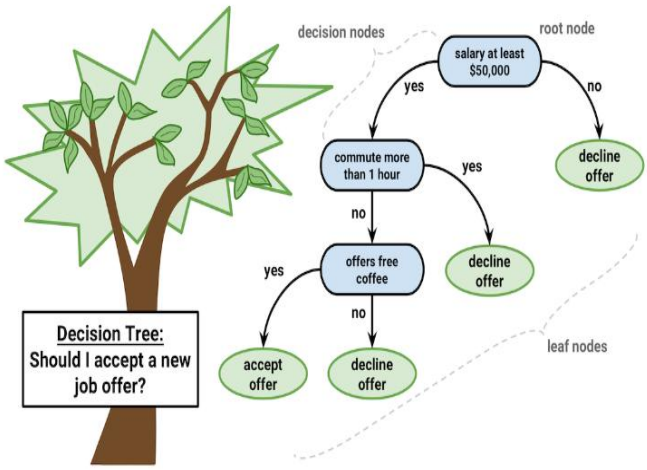
department_marketing	department_technology
0	0
1	0
0	1

Thank You!

Study Objectives

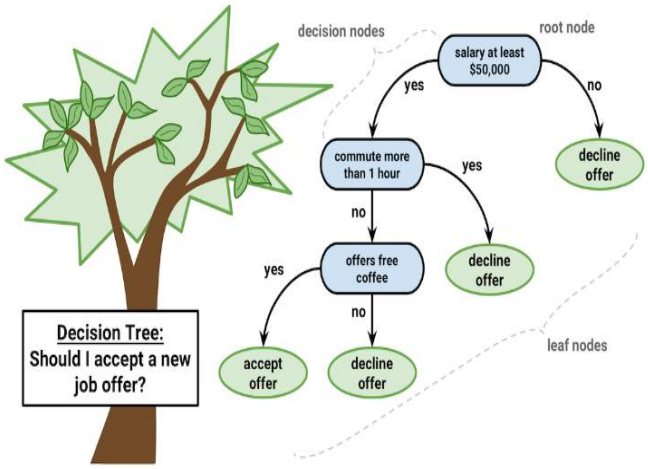
- Interpret and explain decisions
- Explain the process of divine and conquer strategy
- Build a decision tree using the recursive partitioning (rpart) package
- Explain the advantages and disadvantages of a decision tree model
- Build a regression tree and explain the preprocessing steps
- Explain the difference between decision trees and regression trees
- Evaluate the performance of decision and regression tree models

Decision Trees



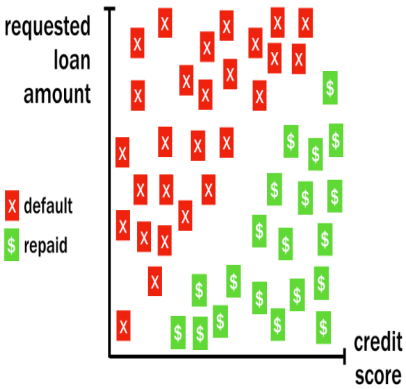
- Hierarchical structure with nodes and directed edges.
- The node at the top is called the root node. The nodes at the bottom are called the leaf nodes or terminal nodes.
- Nodes that are neither the root node or the leaf nodes are called internal/decision nodes. The root and internal nodes have binary test conditions associated with them and each leaf node has an associated class label.

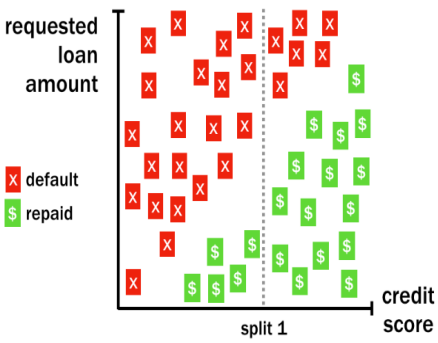
Decision Trees

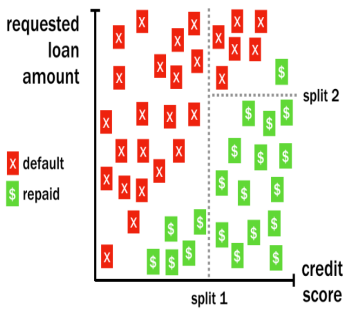


- The goal of a decision tree is to model the relationship between predictors and an outcome of interest.
- Beginning at the root node, data flows through if/else decision nodes that split the data according to its attributes.
- The branches indicate the potential choices, and the leaf nodes denote the final decisions.
- These are also known as terminal nodes because they terminate the decision-making process.

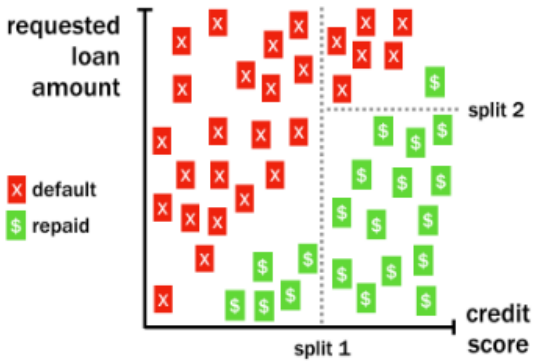
Decision Tree
- Divide and Conquer Process

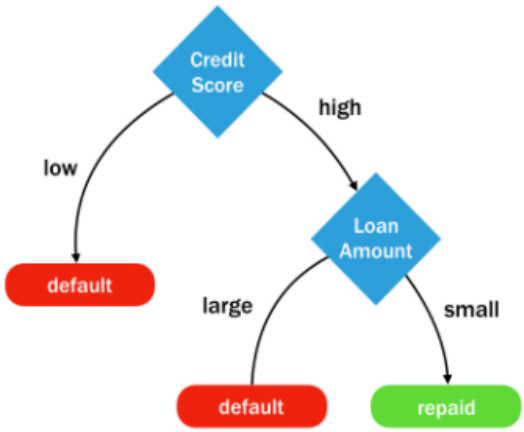






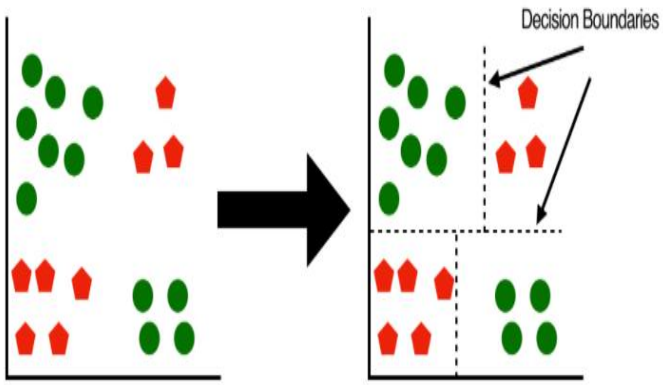
Decision Tree
- Divide and Conquer Process



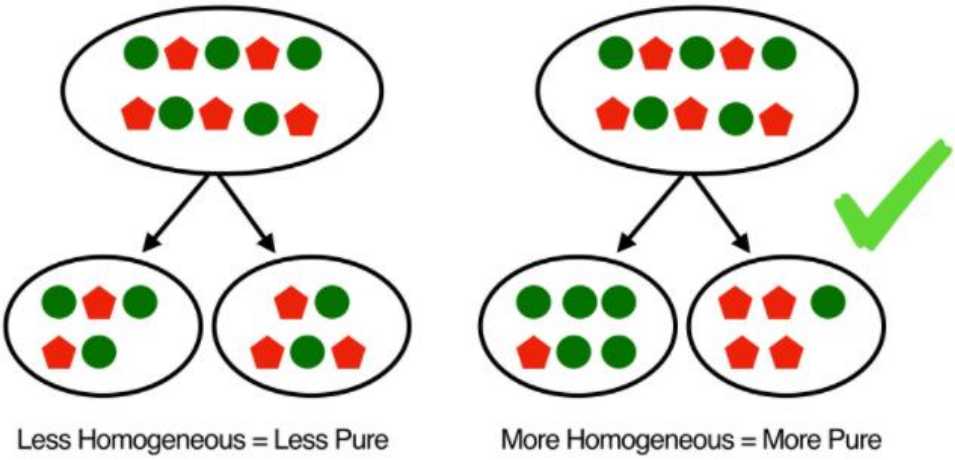


Splitting Data into “Pure” Regions

- The idea behind classification trees is to split the data into subsets where each subset belongs to only one class.
- This is accomplished by dividing the input space into "pure" regions, that is -- regions with samples from only one class.
- With real data, completely pure regions may not be possible, so the decision tree will do the best it can to create regions that are as pure as possible.

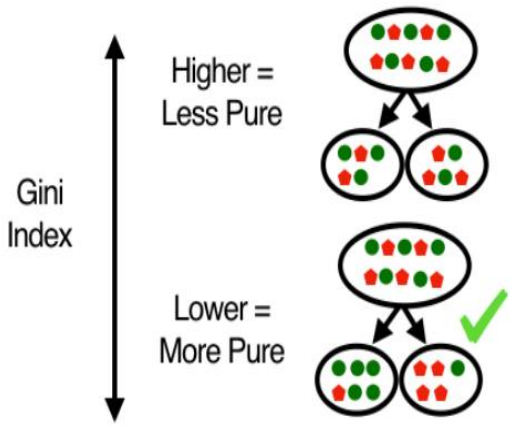


How to Determine Best Splits



Impurity Measure - GINI Index

- A common impurity measure used for determining the best split is the **Gini Index**.
- The lower the Gini Index the higher the purity of the split. So the decision tree will select the split that minimizes the Gini Index.



Computation of GINI Index

$$GINI(t) = 1 - \sum_j [p(j \mid t)]^2$$

C1	**0**
C2	**6**

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1
Gini = 1 – P(C1)² – P(C2)² = 1 – 0 – 1 = 0

C1	**1**
C2	**5**

P(C1) = 1/6 P(C2) = 5/6
Gini = 1 – (1/6)² – (5/6)² = 0.278

C1	**2**
C2	**4**

P(C1) = 2/6 P(C2) = 4/6
Gini = 1 – (2/6)² – (4/6)² = 0.444

Computation of GINI Index -Splitting Based on GINI

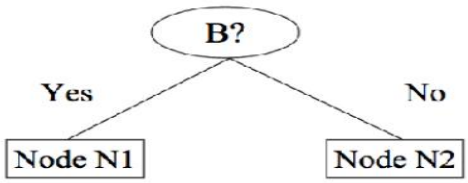
- When a node p is split into k partitions (children), the quality of split is computed as,

$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

where, n_i = number of records at child i,
n = number of records at node p.

Computation of GINI Index -Splitting Based on GINI

- Splits into two partitions
- Effect of Weighing partitions:
  - Larger and Purer Partitions are sought for.



	Parent
C1	6
C2	6
Gini = 0.500	

Gini(N1)
= 1 – (5/6)² – (2/6)²
= 0.194

Gini(N2)
= 1 – (1/6)² – (4/6)²
= 0.528

	N1	N2
C1	5	1
C2	2	4
Gini=0.333		

Gini(Children)
= 7/12 * 0.194 +
5/12 * 0.528
= 0.333

Advantages of Decision Trees

- Training and prediction process is easy to explain.
- Trees can be displayed visually, and are easily interpreted even by a non-expert.
- Decision trees require no normalisation or standardisation for numeric features.
- Trees can also handle categorical features without the need to create dummy binary indicator variables (or to use other types of categorical encodings).

Advantages of Decision Trees

- Decision trees recursively partition the feature space into subsets which allows the decision boundaries to be more complex than a simple linear method.
- It's relatively fast to train a decision tree, so tree methods can handle big datasets.

Disadvantages of Decision Trees

- Large, deep, trees are hard to interpret
- One of the major problems with trees is that they have high variance. A small change in the data can result in a very different series of splits (also making interpretation somewhat precarious).
- High variance leads to poor model performance. If not tuned properly, it's easy to create overly-complex trees that do not generalise well to new data. This is also known as "overfitting".

Training Decision Trees in R

- Classification Trees – ctree package
- Classification & Regression Trees – rpart package

Building Trees in R

- Simply use the rpart function with the R formula interface to specify the outcome and predictors.
- The "class" parameter tells rpart to build a classification tree. And like the other machine learning methods, the predict function obtains the predicted class values for the test dataset.

```
# building a simple rpart classification tree
library(rpart)
m <- rpart(outcome ~ loan_amount + credit_score, data = loans,
           method = "class")
```

```
# making predictions from an rpart tree
p <- predict(m, test_data, type = "class")
```

Hyperparameters in Decision Trees

- There are three important parameters that affect the size and complexity of our tree:
  - minsplit parameter
  - cost complexity parameter
  - maxdepth parameter

Hyperparameters in Decision Trees

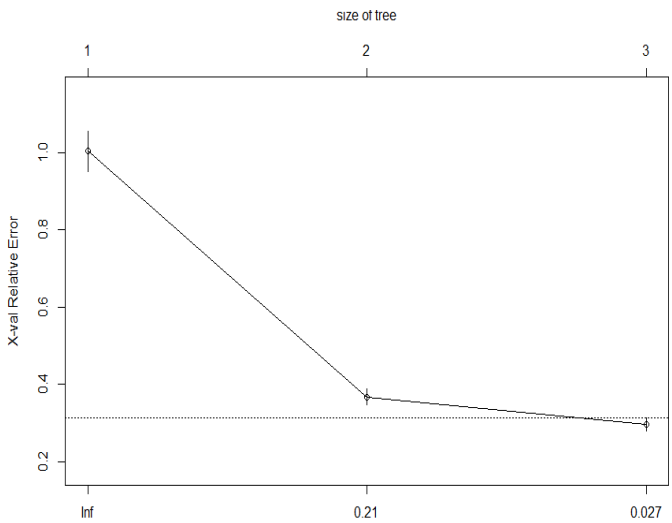
- *Minsplit*
  - The minsplit parameter defines the minimum number of data points that are needed in order for the algorithm to attempt a split before it is forced to create a leaf node. The default value is 20.
- *Cost Complexity Parameter*
  - The cp parameter is the complexity parameter and the default value of this is 0.1. It controls the trade-off between model accuracy and model simplicity by penalising tree size By penalising complexity, it avoids splits that only improve training accuracy slightly.
- *Maxdepth*
  - The maxdepth parameter limits the maximum number of nodes between a leaf node and the root node. The default value of 30, thus, allowing for fairly large trees to be built.

Cost-Complexity Parameter (CP)

- The CP serves as a penalty term to control tree size, and is always monotonic with the number of splits (nsplit).
- The smaller the value of CP, the more complex will be the tree (the greater the number of splits).

Cost-Complexity Parameter (CP)

- The rpart() function computes the 10-fold cross-validated error of the model over various values for CP and stores the results in a table inside the model.
- We can also very easily plot the cross-validated error across the different values of CP using the plotcp() function. This makes it easy to see right away what the optimal value for CP is.



Cost-Complexity Parameter (CP)

- The optimal CP value can be generated using the CP_table. The CP for the least xerror column is chosen as the optimal cp value.
- The tree is then pruned using the optimized CP value

```
> print(rtmodel$cptable)
        CP nsplit rel error    xerror       xstd
1 0.6333406      0 1.0000000 1.0029386 0.05261076
2 0.0718027      1 0.3666594 0.3675503 0.02112518
3 0.0100000      2 0.2948567 0.2960937 0.01731306
```

Regression Trees

- In regression, the goal is to predict a numeric outcome from a set of inputs. The dependent variable, or "response", can be either continuous or integer-valued.
- Regression trees are built in much the same way as they are for classification.

- Beginning at the root node, the data is partitioned using a divide-and-conquer strategy according to the feature that will result in the greatest increase in homogeneity in the outcome after a split is performed.

Regression Trees

- For numeric decision trees, homogeneity is measured by statistics such as variance, standard deviation, or absolute deviation from the mean.

Common Metrics for Regression

- There are several metrics for regression but the two popular ones are:
  - *Mean Absolute Error (MAE)*
  - *Root Mean Square Error (RMSE)*

Common Metrics for Regression

- MAE is the average absolute distance between the actual (or observed values) and predicted values.
- The RMSE is similar, but instead of taking the absolute difference between errors, you square the differences of the errors. Then you'll average those values across your test set and take the square root of the whole thing.

- Mean Absolute Error (MAE)

$$MAE = \frac{1}{n}\sum |\, actual - predicted \,|$$

- Root Mean Square Error (RMSE)

$$RMSE = \sqrt{\frac{1}{n}\sum (actual - predicted)^2}$$

Final Thoughts…

- The idea behind classification trees is to split the data into subsets where each subset belongs to only one class.
- A decision tree comprises a root node, decision nodes and leaf nodes.
- The Gini Index is used to determine the purity of the regions for a split.
- Hyperparameters controls such as minsplit, maxdepth, and cost complexity parameter can be used to prune a complex decision tree
- The goal of a regression tree is the prediction of a numeric outcome
- Evaluation of a regression tree model include Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE).

Thank You!

Unit 4
Random Forest

Nana Boateng

Study Objectives

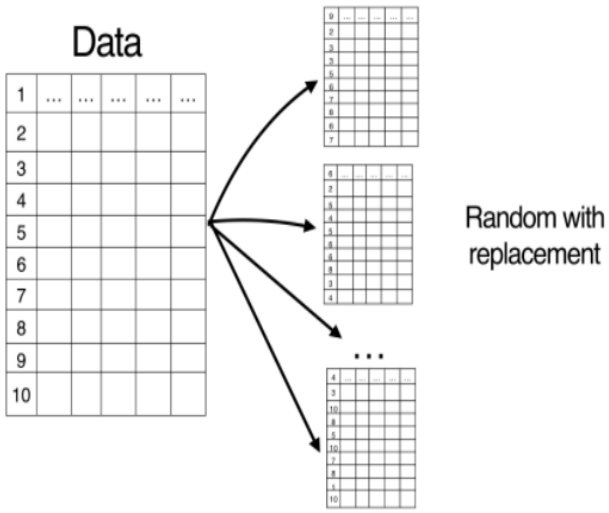
- After completing this unit, students should be able to:
- Explain how Random Forest applies the concept of bagging
- Explain what is meant by Out Of Bag (OOB) Error
- Explain the advantages and disadvantages of random forest

Bagging Trees

- Often a small change in the data can result in a very different series of splits, which can also make model interpretation somewhat precarious.
- Bagging, and in particular, bagged trees, averages many trees to reduce this variance.
- Combining several models into one is what's called an **ensemble model** and averaging is one of the easiest ways to create an ensemble from a collection of models.

Bagging Trees

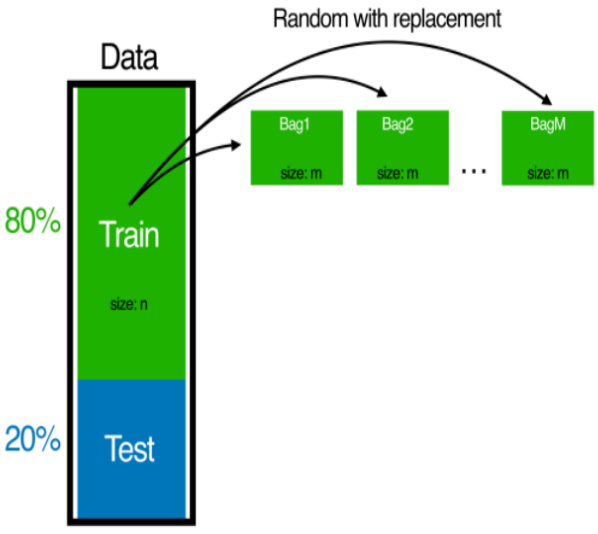
- Bagging is a ensemble method and the term "bagging" is shorthand for **bootstrap aggregation**.
- Bagging uses bootstrap sampling and aggregates the individual models by averaging.
- Bootstrap sampling, or "bootstrapping", simply means sampling rows at random from the training dataset, with replacement.



Steps to Bagging

Step 1

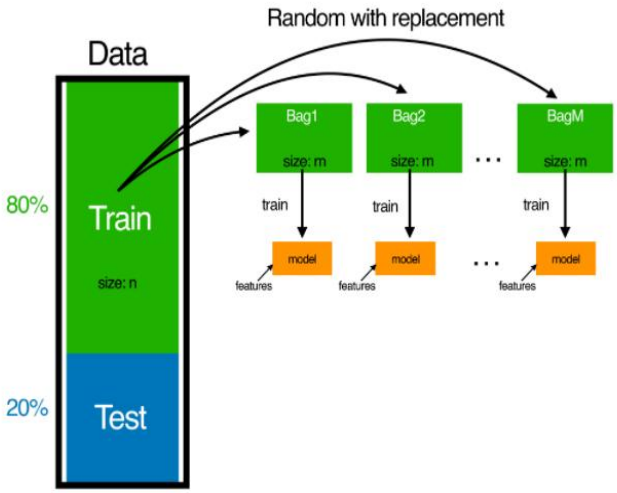
- Draw B samples with replacement from the original training set, where B is a number less than or equal to the N, number of total samples in the training set.
- With bagged trees, a common choice for B is one-half of N.



Steps to Bagging

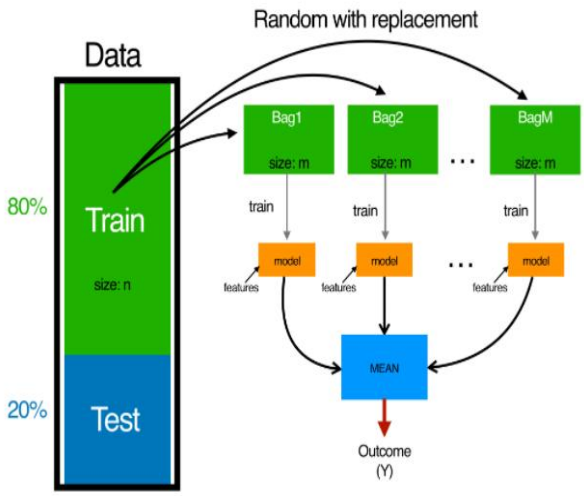
Step 2

- Train a decision tree on the newly created bootstrapped sample. Repeat steps 1 through 2 for as many times as you like -- that could be 20 times, 100 times or 1000.
- Typically, the more trees, the better the model.



Bagging Outcome

- The bagged, or "ensemble" prediction is the average prediction across the bootstrapped trees.
- Bagging can dramatically reduce the variance of unstable models such as trees.
- This results in a better performing model.



Bootstrap Example

- Original data (random sample):
  - {1,2,3,1,2,1,1} (n = 7)
- Bootstrap random sample data:
  - {1,2,1,2,1,1,3,} (n = 7); mean = 1.571
  - {1,1,2,2,3,1,1} (n = 7); mean = 1.571
  - {1,2,1,2,1,1,2} (n = 7); mean = 1.428

- The estimated mean for our original data is the mean of the statistic for each bootstrap sample (1.571+1.571+1.428)/3 = ~1.522

Flaws of Bootstrapping

- The problem with Bagging algorithm is it uses uses Gini-Index, a greedy algorithm to find the best split.
- So we end up with trees that are structurally similar to each other. The trees are highly correlated among the predictions.

- The issue with high correlation is addressed by the Random Forest Algorithm

Random Forest

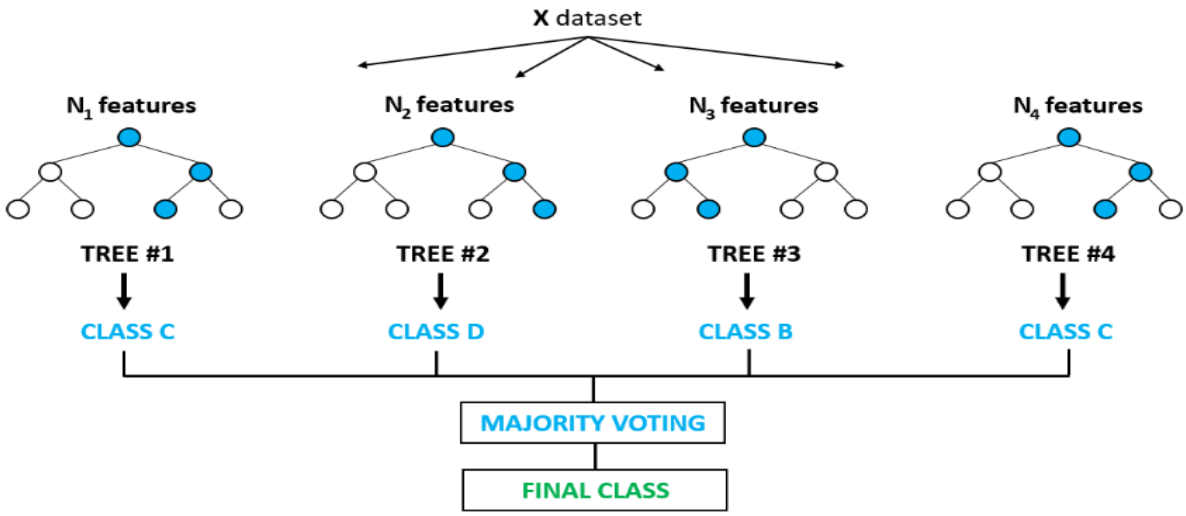
- The basic idea behind random forests is identical to bagging -- both are ensembles of trees trained on bootstrapped samples of the training data.

- However, in the Random Forest algorithm, there is slight tweak to the way the decision trees are built that leads to better performance.

Random Forest

- The key difference is that when training the trees that make up the ensemble, we add a bit of extra randomness to the model -- hence the name, Random Forests.
- At each split in the tree, rather than considering all features, or input variables, for the split, we sample a subset of these features and consider only these few variables as a candidates for the split.
- The technique of sampling variables from the input or feature space is also feature bagging, or the random subspace method.
- Random Forests improve upon bagging by reducing the correlation between the sampled trees.

Random Forest



Random Forest

- Better Performance
- Sample subset of features
- Improved version of bagging
- Reduced Correlation between sampled trees

Random Forest Algorithm in R

- To run the Random Forest algorithm you will use the randomForest function from the randomForest package

```
library (randomForest)
?randomForest
```

Classification and Regression with Random Forest

Description

`randomForest` implements Breiman's random forest algorithm (based on Breiman and Cutler's original Fortran code) for classification and regression. It can also be used in unsupervised mode for assessing proximities among data points.

Usage

```
## S3 method for class 'formula'
randomForest(formula, data=NULL, ..., subset, na.action=na.fail)
## Default S3 method:
randomForest(x, y=NULL,  xtest=NULL, ytest=NULL, ntree=500,
             mtry=if (!is.null(y) && !is.factor(y))
             max(floor(ncol(x)/3), 1) else floor(sqrt(ncol(x))),
             replace=TRUE, classwt=NULL, cutoff, strata,
             sampsize = if (replace) nrow(x) else ceiling(.632*nrow(x)),
             nodesize = if (!is.null(y) && !is.factor(y)) 5 else 1,
             maxnodes = NULL,
             importance=FALSE, localImp=FALSE, nPerm=1,
             proximity, oob.prox=proximity,
             norm.votes=TRUE, do.trace=FALSE,
             keep.forest=!is.null(y) && is.null(xtest), corr.bias=FALSE,
             keep.inbag=FALSE, ...)
## S3 method for class 'randomForest'
print(x, ...)
```


Advantages of Random Forest Package

- Random Forest is based on the bagging algorithm and uses Ensemble Learning technique. It creates as many trees on the subset of the data and combines the output of all the trees. In this way it reduces overfitting problem in decision trees and also reduces the variance and therefore improves the accuracy.
- Random Forest can be used to solve both classification as well as regression problems.
- Random Forest works well with both categorical and continuous variables.
- No feature scaling required: No feature scaling (standardization and normalization) required in case of Random Forest as it uses rule-based approach instead of distance calculation.
- Random Forest algorithm is very stable. Even if a new data point is introduced in the dataset, the overall algorithm is not affected much since the new data may impact one tree, but it is very hard for it to impact all the trees.

Disadvantage of the Random Forest Package

- Complexity: Random Forest creates a lot of trees (unlike only one tree in case of decision tree) and combines their outputs. On the other hand decision tree is simple and does not require so much computational resources.

- Longer Training Period: Random Forest require much more time to train as compared to decision trees as it generates a lot of trees (instead of one tree in case of decision tree) and makes decision on the majority of votes.

Unit 5 – K-Nearest Neighbour (kNN) Algorithm

Nana Boateng

Study Objectives

After completing this unit, students should be able:

- Explain the kNN approach to classification
- Explain the concept of similarity using Euclidian Distance
- Understand the requirements of kNN in R
- Build a kNN model

K Nearest Neighbour (kNN)

- The nearest neighbours approach to classification is utilised by the kNN algorithm.

- The kNN algorithm takes as input a training dataset comprising instances classified into several categories
  - labelled by a nominal variable.

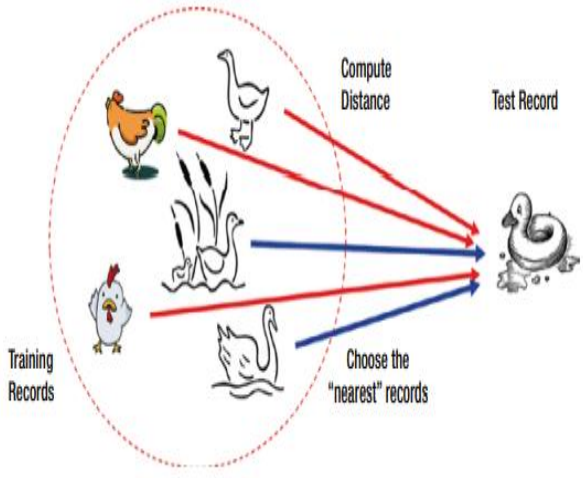
K Nearest Neighbour (kNN)

- Assume that we have a test dataset containing unlabelled instances that otherwise have the same features as the training data.

- For each instance in the test dataset, kNN identifies *k* records in the training data that are the "nearest" in similarity, where *k* is an integer specified in advance.
- The unlabelled test instance is assigned the class of the majority of the k nearest neighbours.

How kNN Works

- The unknown sample is assigned the nearest class among the k-nearest neighbours pattern.
- The idea is to look for the records that are similar to, or “near,” the record to be classified in the training records that have values close to X = (x1 , x2 , x3 , …).
- These records are grouped into classes based on the “closeness,” and the unknown sample will look for the class (defined by k) and identifies itself to that class that is nearest in the k-space.



How kNN - Distance

- The “closeness” is defined in terms of Euclidean distance. Euclidean distance is specified by the following formula:
  - p and q are the examples to be compared
  - each has n attributes.
  - p1 refers to the value of the first attribute of instance p,
  - q1 refers to the value of the first attribute of instance q.

$$dist\ (p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots\ldots + (pn - qn)^2}$$

The kNN Algorithm - Distance

- Assume a tomato is [sweetness = 6, crunchiness = 4] and a green bean [sweetness = 3, crunchiness = 7], then we can use the formula as follows:

- $dist\ (tomato, green\ bean) = \sqrt{(6-3)^2 + (4-7)^2}$ = 4.24

The kNN Algorithm - Distance

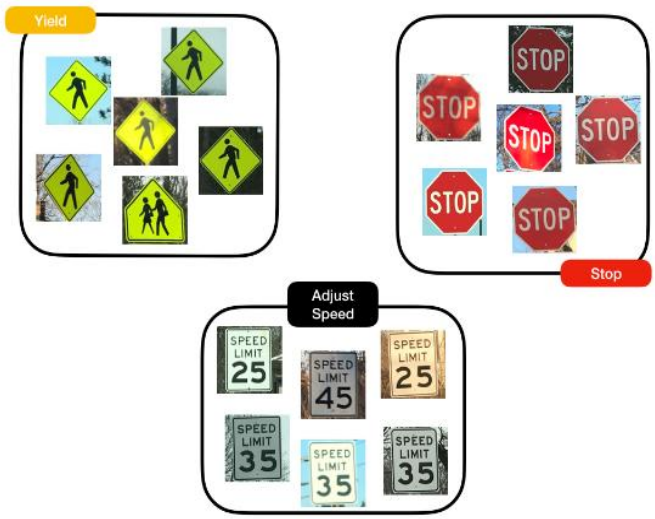
- Lets assume these values for other instances and calculate their distance to our instance of tomato:

ingredient	sweetness	crunchiness	food type	distance to tomato
grape	8	5	fruit	sqrt((6-8)^2 + (4-5)^2) = 2.2
green bean	3	7	vegetable	sqrt((6-3)^2 + (4-7)^2) = 4.2
nuts	3	6	protein	sqrt((6-3)^2 + (4-6)^2) = 3.6
orange	7	3	fruit	sqrt((6-7)^2 + (4-3)^2) = 1.4

- To classify the tomato as a vegetable, protein, or fruit, we'll begin by assigning the tomato, the food type of its nearest neighbour.
- The orange is the nearest neighbour to the tomato, with a distance of 1.4.

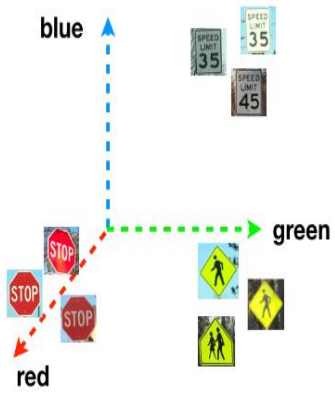
Illustrative Application of KNN - Self Driving Car

- To begin training a self-driving car, you might guide it by showing the desired actions for each type of traffic sign.
- For example, you STOP at intersections, YIELD to pedestrians, and adjust SPEED appropriately. Over time, as you instruct it, the vehicle creates a database that logs the signs along with the corresponding target behaviour.



Measuring Similarity

- By imagining the colour as a 3-dimensional feature space measuring levels of red, green, and blue, signs of similar colour are located naturally close to one another.
- Once the feature space has been constructed in this way, you can measure distance using Euclidean distance.



$$\text{dist}(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2}$$

Applying KNN in R

- The knn function is part of the class package, and requires three parameters: first, the set of training data; second, the test data to be classified; and third, the labels for the training data.

```
##KNN Algorithm
pred <- knn(train_data, test_data, train_labels)
```

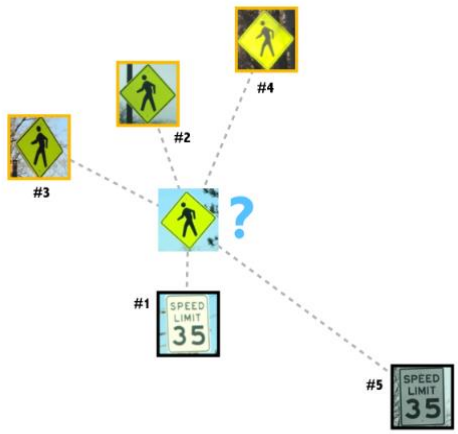
What is the ‘k’in kNN

- The letter k is a variable that specifies the number of neighbours to consider when making the classification.
- You can imagine it as determining the size of the neighborhoods.
- R has uses a default value of ‘1’ for ‘k. This means that only the single nearest, most similar, neighboúr will be used to classify the unlabelled data.

What is the ‘k’in kNN…..cont’d

- Suppose our vehicle observed the sign at the center in figure 2. Its five nearest neighbours are depicted.
- The single nearest neighbour is a speed limit sign, which shares a very similar background colour. Unfortunately, in this case, a kNN classifier with k set to one would make an incorrect classification of SPEED.

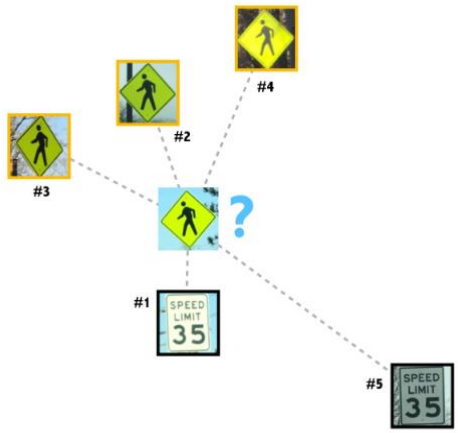
- *Figure 2: Road Signs*



What is the ‘k’in kNN…..cont’d

- Slightly further away are the 2nd and 3rd nearest neighbors, which are all pedestrian crossing signs.
- Suppose we set k to three. What would happen? The three nearest neighbours, a speed limit sign and two pedestrian crossing signs, would take a vote. The category with the majority of nearest neighbours, in this case the pedestrian crossing sign wins.
- Increasing k to 5 allows the five nearest neighbours to vote. The pedestrian crossing sign still wins with a margin of 3:2 Note that in the case of a tie, the winner is typically decided at random.

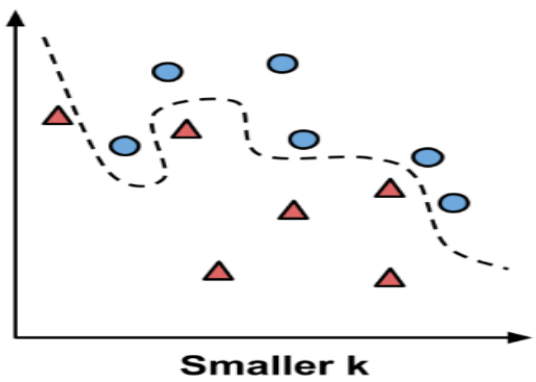
- *Figure 2: Road Signs*



Bigger ‘k’is not always better

- In the previous example, setting k to a higher value resulted in a correct prediction. But it is not always the case that bigger ‘k’ is better.
- A small k creates very small neighborhoods; the classifier is able to discover very subtle patterns. As figure 3 illustrates, you might imagine it as being able to distinguish between groups even when their boundary is somewhat "fuzzy."

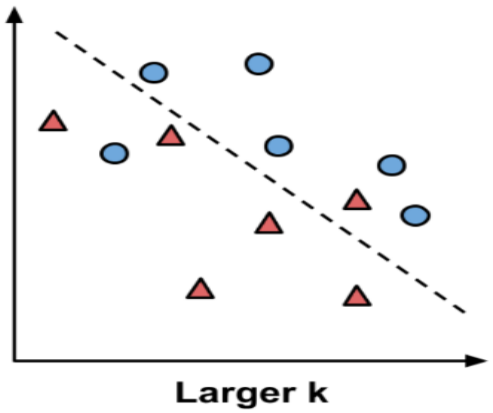
- *Figure 3: Smaller k*



Bigger ‘k’is not always better…

- On the other hand, sometimes a "fuzzy" boundary is not a true pattern, but rather due to some other factor that adds randomness into the data. This is called noise. Setting k larger, as this image shows, ignores some potentially-noisy points in an effort to discover a broader, more general pattern.

- *Figure 4: Larger ‘k’*



The kNN Algorithm – pragmatically selecting k

- In practice, choosing k depends on the difficulty of the concept to be learned and the number of instances in the training data.
  - A common practice is to set k equal to the square root of the number of training instances.

- May not always result in the single best k.

- Alternative approaches:
  - to test several k values on a variety of test datasets and choose the one that delivers the best classification performance.
  - Increase the size of the training set: unless the data is very noisy, larger and more representative training datasets can make the choice of k less important
  - This is because even subtle concepts will have a sufficiently large pool of instances to vote as nearest neighbours.

Preprocessing Data for Nearest Neighbours -Dummy Encoding

- kNN assumes numerical data. Categorical variables need to be dummy encoded (0/1).
- Nearest neighbor learners use distance functions to identify the most similar for nearest examples. Many common distance functions assume that your data is in numeric format, as it is difficult to define the distance between categories.

rectangle = 1

diamond = 0

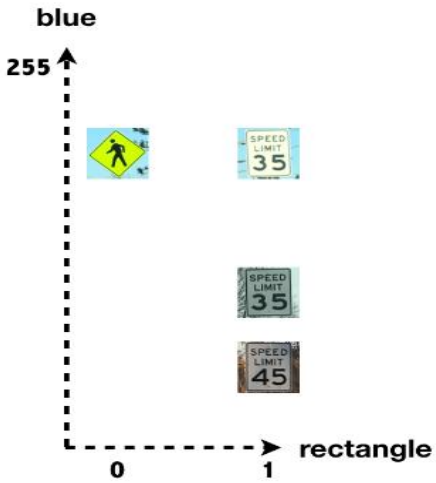
rectangle = 0

diamond = 1

Preparing Data for Nearest Neighbours - Normalisation

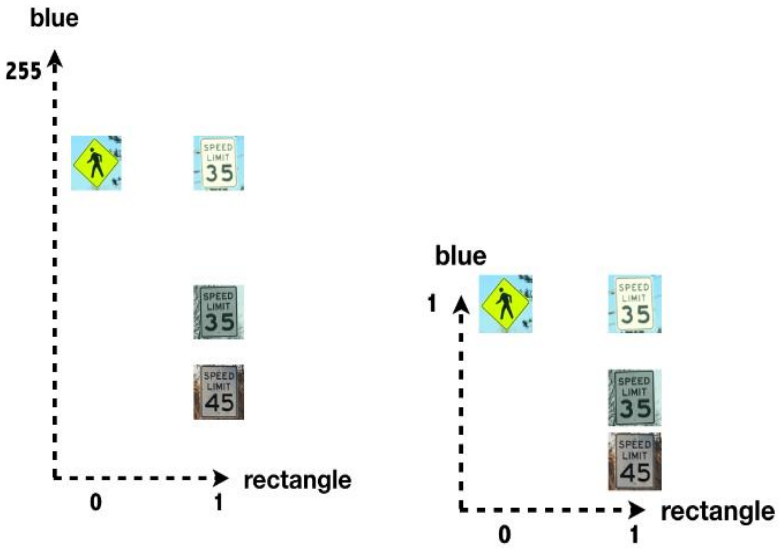
- In figure 5, each colour component ranges from a minimum of zero to a maximum of 255.
- However, suppose that we added the 1/0 dummy variables for sign shapes into the distance calculation. Two different shapes may differ by at most one unit, but two different colors may differ by as much as 255 units!
- Such a different scale allows the features with a wider range to have more influence over the distance calculation, as figure 5 illustrates. Here, the one of the topmost speed limit sign is closer to the pedestrian sign than it is to its correct neighbours, simply because the range of blue values is wider than the 0-to-1 range of shape values.

Figure 5: Range of Colour Component & Shape



Preparing Data for Nearest Neighbours - Normalisation

- Compressing the blue axis so that it also follows a 0-to-1 range corrects this issue, and the speed limit sign is now closer to its true neighborhood.



Requirements for kNN in R

- Generally, k gets decided based on the square root of data points
- Dummy Encoding
- Data Normalisation
- Installation of “class” package

Naïve Bayes

Nana Boateng

Study Objectives

- After completing this unit, students should be able to:
- Explain the components of the bayes theorem
- Explain the concept of conditional probability
- Apply the naïve bayes algorithm
- Make predictions of outcomes based on conditional probabilities

What is Naïve Bayes?

- The Naïve Bayes classifier is one of most popular classifier used in the machine-learning. The Naïve Bayes classifier is a simple probabilistic classifier based on Bayes' theorem.

- It is based on the idea that the predictor variables in a machine learning model are independent of each other. This means that the model's outcome depends on a set of independent variables that do not influence one another.

Why Naïve?

- The model is referred to as 'naïve' because in real-world scenarios, predictor variables are often correlated with one another and rarely completely independent.

- Even though Naïve Bayes is a simple technique, it provides good performance in many complex real-world problems.

Use Cases of Naïve Bayes

- Use cases of Naïve Bayes include:
  - Spam Filtering
  - Text Classification
  - Document Categorisation
  - Sentiment Analysis
  - Recommendation Systems
  - Real-time Prediction
  - Medical Diagnosis
  - Image Recognition
  - Credit Scoring
  - Anomaly Detection

How the Naïve Bayes Algorithm Works

- The principle behind Naive Bayes is the Bayes theorem also known as the Bayes Rule.

- It involves the use of probabilistic classification based on the approximating joint distribution with an assumption of independence, and then decomposing this probability into a product of conditional probability.

Bayes Theorem

- Mathematically, the Bayes Theorem is represented as:

- $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$

Where:
- P(A|B): Conditional probability of event A occurring, given the event B
- P(A): Probability of event A occurring
- P(B): Probability of event B occurring
- P(B|A): Conditional probability of event B occurring, given the event A

Bayes Theorem -Terminology

- A is referred to as the proposition and B is the evidence
- P(A) represents the prior probability of the proposition
- P(B) represents the prior probability of evidence
- P(A|B) is referred to as the posterior
- P(B|A) is referred to as the likelihood

- $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$

Bayes Theorem

- The formula:
- $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$
- Can be seen as:
- $Posterior = \frac{Liklihood * Proposition\ Prior\ Probabiity}{Evidence}$

Naïve Bayes Algorithm

- In the context of classification, Bayes theorem provides a formula for calculating the probability of a given record belonging to a class for the given set of record attributes.
- Suppose you have m classes, C1, C2, C3, ... Cm and you know the probability of classes P(C1), P(C2), ... P(Cm). If you know the probability of occurrence of x1, x2, x3, ... attributes within each class, then by using Bayes' theorem, you can calculate the probability that the record belongs to class Ci:

$$P(Ci|X_1,X_2,X_3,\ldots X_p) = \frac{P(X_1,X_2,X_3,\ldots X_p|C_i)P(C_i)}{P(X_1,X_2,\ldots X_p|C_1)+P(X_1,X_2,\ldots X_p|C_2)\ldots\ldots P(X_1,X_2,..X_p|C_m)}$$

Naïve Bayes Algorithm

- P(Ci) is the prior probability of belonging to class Ci in the absence of any other attributes. (Ci|Xi) is the posterior probability of Xi belonging to class Ci.
- In order to classify a record using Bayes' theorem, you compute its chance of belonging to each class Ci and then classify based on the highest probability score calculated using the preceding formula

Naïve Bayes Algorithm

- It would be extremely computationally expensive to compute P(X|Ci) for data sets with many attributes. Hence, a naïve assumption is made that presumes that each record is independent of the others, given the class label of the sample.
- It is reasonable to assume that the predictor attributes are all mutually independent within each class, so we can simplify the equation:

$$P(X|C_i) = \prod_{k=1}^{n} P(X_k|C_i)$$

- Assume that each data sample is represented by an n-dimensional vector, X = {x1, x2, ... xn}, samples from n attributes of categorical class values A1, A2, ... And with m classes C1, C2, ... Cm, then P(Xk|Ci) = n/N, where n is the number of training samples of class Ci having value Xk for Ak, and N is the total number of training samples belonging to Ci.

Illustrative Example – Naïve Bayes

Sample Training Set

ID	Purchase Frequency	Credit Rating	Age	Class Label: Yes = Approved No = Denied
1	Medium	OK	< 35	No
2	Medium	Excellent	< 35	No
3	High	Fair	35–40	Yes
4	High	Fair	> 40	Yes
5	Low	Excellent	> 40	Yes
6	Low	OK	> 40	No
7	Low	Excellent	35–40	Yes
8	Medium	Fair	< 35	No
9	Low	Fair	< 35	No
10	Medium	Excellent	> 40	No
11	High	Fair	< 35	Yes
12	Medium	Excellent	35–40	No
13	Medium	Fair	35–40	Yes
14	High	OK	< 35	No

- The data samples in this training set have the attributes Age, Purchase Frequency, and Credit Rating.
- The class label attribute has two distinct classes: Approved or Denied. Let C1 correspond to the class Approved, and C2 correspond to class Denied.
- Using the Naïve Bayes classifier, we want to classify an unknown label sample X:
- X = (Age >40, Purchase Frequency = Medium, Credit Rating = Excellent)

Illustrative Example….Continued

- P(Application Approval = Yes) = 6/14 = 0.428
- P(Application Approval = No) = 8/14 = 0.571
- Let's compute P(X|Ci), for i =1, 2, ... conditional probabilities:
- P(Age > 40 | Approval = Yes) = 2/6 = 0.333
- P(Age > 40 | Approval = No) = 2/8 = 0.25
- P(Purchase Frequency = Medium | Approval = Yes) =1/6 = 0.167
- P(Purchase Frequency = Medium | Approval = No) = 5/8 = 0.625
- P(Credit Rating = Excellent | Approval = Yes) = 2/6 = 0.333
- P(Credit Rating = Excellent | Approval = No) = 3/8 = 0.375
- Using these probabilities, we can obtain the following:
- P(X | Approval = Yes) = 0.333 × 0.167 × 0.333 = 0.0185
- P(X | Approval = No) = 0.25 × 0.625 × 0.375 = 0.0586
- P(X | Approval = Yes) × P(Approval = Yes) = 0.0185 × 0.428 = **0.0079**
- P(X | Approval = No) × P(Approval = No) = 0.0586 × 0.571= **0.03346 (choose the classification with the highest probability)**
- **The Naïve Bayesian classifier predicts Approval = No for the given set of sample X.**

Sample Training Set

ID	Purchase Frequency	Credit Rating	Age	Class Label: Yes = Approved No = Denied
1	Medium	OK	< 35	No
2	Medium	Excellent	< 35	No
3	High	Fair	35–40	Yes
4	High	Fair	> 40	Yes
5	Low	Excellent	> 40	Yes
6	Low	OK	> 40	No
7	Low	Excellent	35–40	Yes
8	Medium	Fair	< 35	No
9	Low	Fair	< 35	No
10	Medium	Excellent	> 40	No
11	High	Fair	< 35	Yes
12	Medium	Excellent	35–40	No
13	Medium	Fair	35–40	Yes
14	High	OK	< 35	No

Final Thoughts….

- Naïve Bayes is a classification model
- It is a simple probabilistic classifier based on Bayes' theorem.
- Predictors are assumed to be independent of each other
- Classifications are based on conditional probability

Unit 7 – Support Vector Machines (SVM)

Nana Boateng

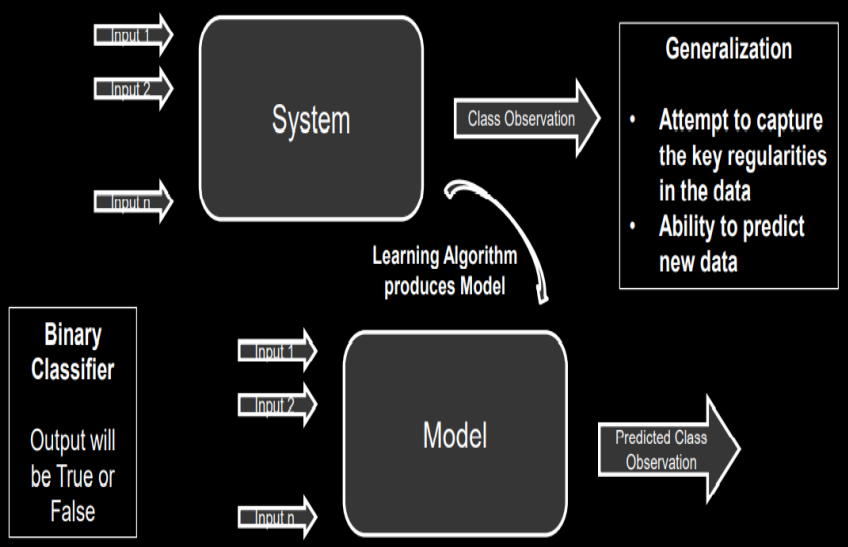
Learning Outcomes

After completing this unit, students should be able to:

- Identify support vectors within a dataset
- Explain a Decision Boundary
- Explain different kernels for SVM
- Create a soft/hard margin
- Distinguish between linearly separable and non-linear separable cases
- Explain the concept of the kernel trick

Support Vector Machine

- Build model of a system
  - Use observations of how the system has behaved given certain inputs
  - Predict how system will behave for a previously unobserved set of input

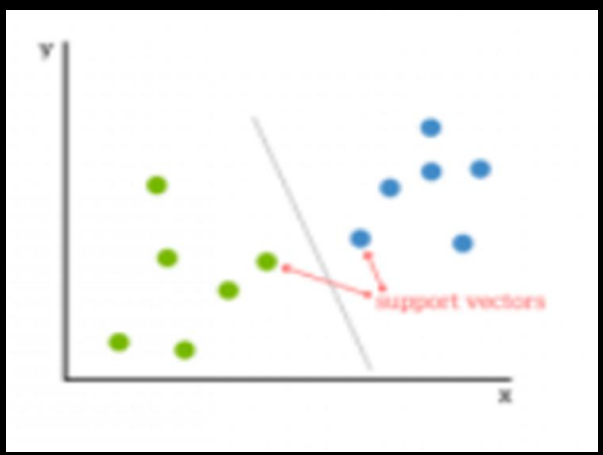


Support Vector Machines

- A Support Vector Machine (SVM) is a supervised machine learning algorithm that can be employed for both classification and regression purposes.
- SVMs are based on the idea of finding a hyperplane that best divides a dataset into classes.

Support Vectors

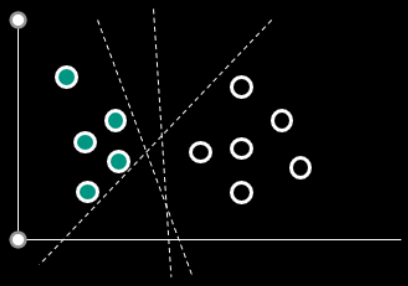
- Support vectors are the data points nearest to the hyperplane, the points of a data set that, if removed, would alter the position of the dividing hyperplane.
- Because of this, they can be considered the critical elements of a data set.



Support Vector Machines

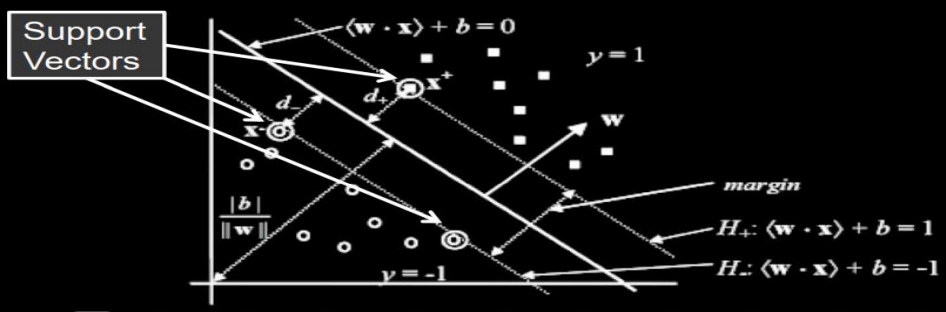
- Support Vector Machines extend the idea associated with a Perceptron to find a certain hyperplane for classification purposes
- SVMs can also handle data instances that are not immediately linearly seperable

How many ways are there to separate the class instances using lines (i.e hyperplanes)?



Support Vector Machines

- SVM – Seperable Case



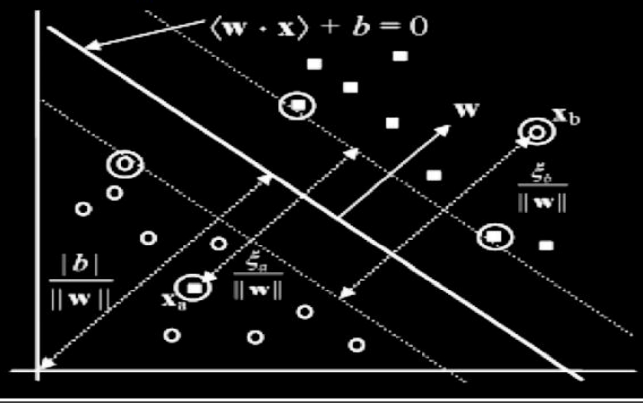
Choosing the maximal margin hyperplane minimizes the upper bound of classification errors

Liu, Web Data Mining

The weight vector w defines a direction perpendicular to the hyperplane.
The SVM maximises the margin between positive and negative data points.
d_+ is the shortest distance from the separating hyperplane to the closest positive data point.
d_- is the shortest distance from the separating hyperplane to the closest positive data point.
The margin is the sum of d_+ and d_-
SVM looks for the hyperplane with the largest margin (the **maximal margin hyperplane**).

Support Vector Machines

- SVM – Non-Seperable Case

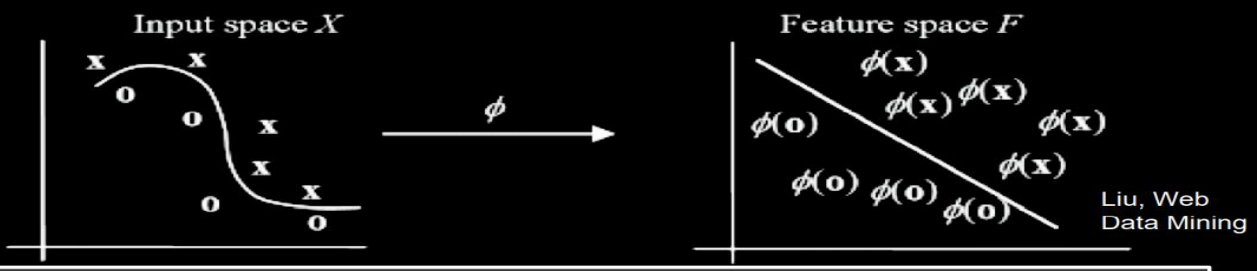


x_a and x_b are error points.

We can account for the error points by integrating penalty calculations into the SVM.

Support Vector Machines

- SVM – Kernel Methods



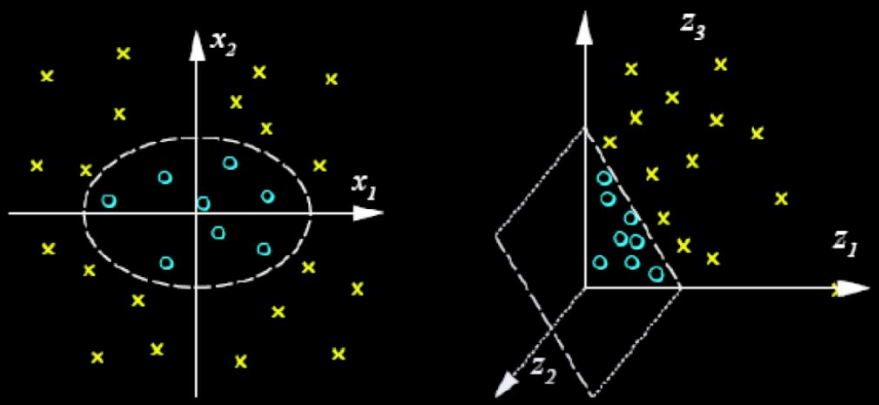
In the Input Space the decision boundary is non-linear. To create a classifier in this case we can use the same techniques as with the linear case – however, we must transform the input data from the original space into another space called a Feature Space F by using a non-linear mapping ϕ.

$\phi: X \rightarrow F$ with $x \longmapsto \phi(x)$

The Feature Space will typically have many more dimensions. The same linear SVM method can then be applied.

Support Vector Machines

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$$



The Kernel Trick

- The basic idea is to devise a mathematical transformation that renders the problem linearly separable.

For Example:

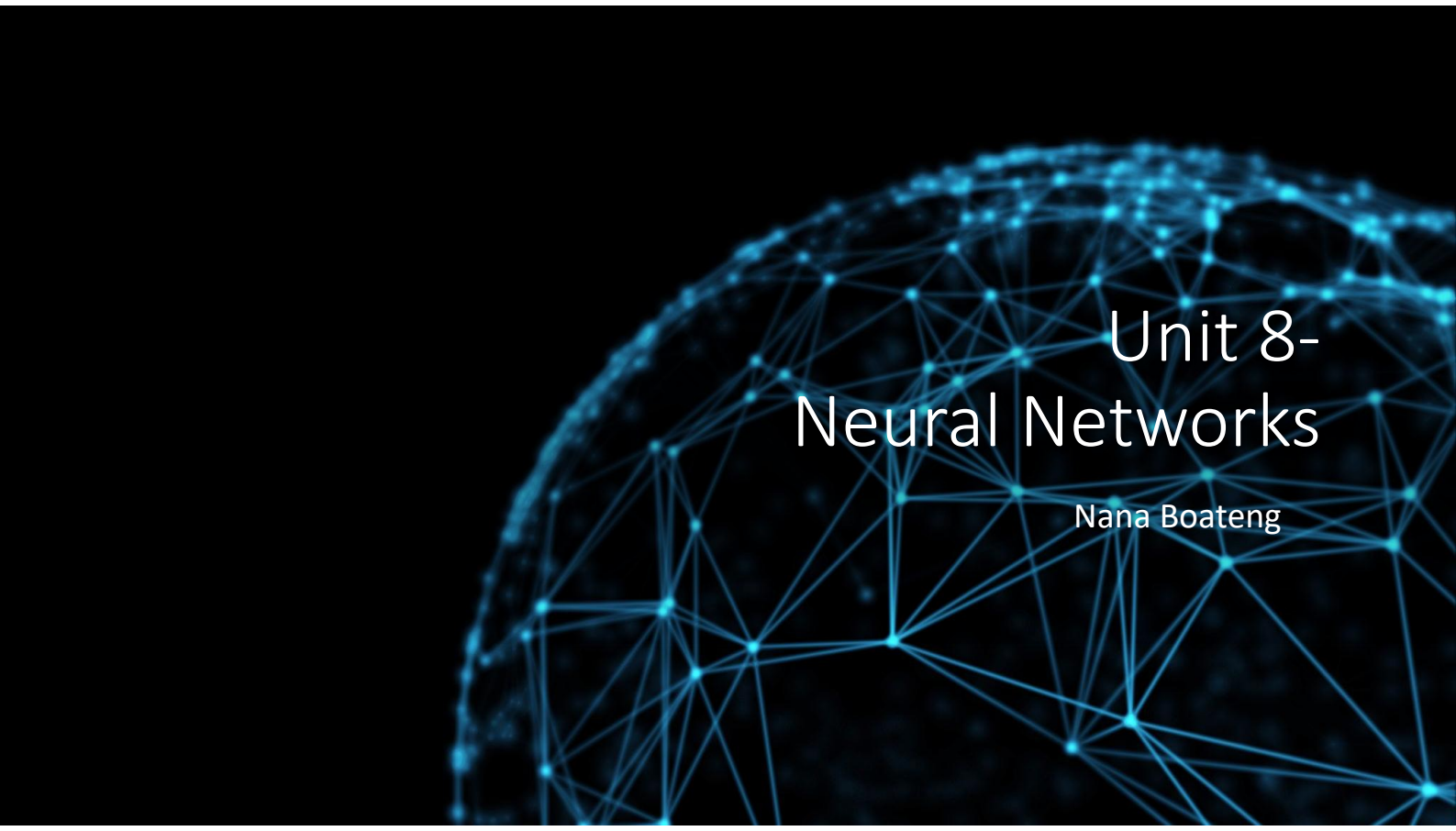
- $x_1^2 + x_2^2 = 0.49$
- Map x_1^2 to a new variable X1, and x_2^2 to X2

The equation of the boundary in the X1 – X2 space becomes:

- X1 + X2 = 0.49 (which is a straight line!)
- For Exmple.

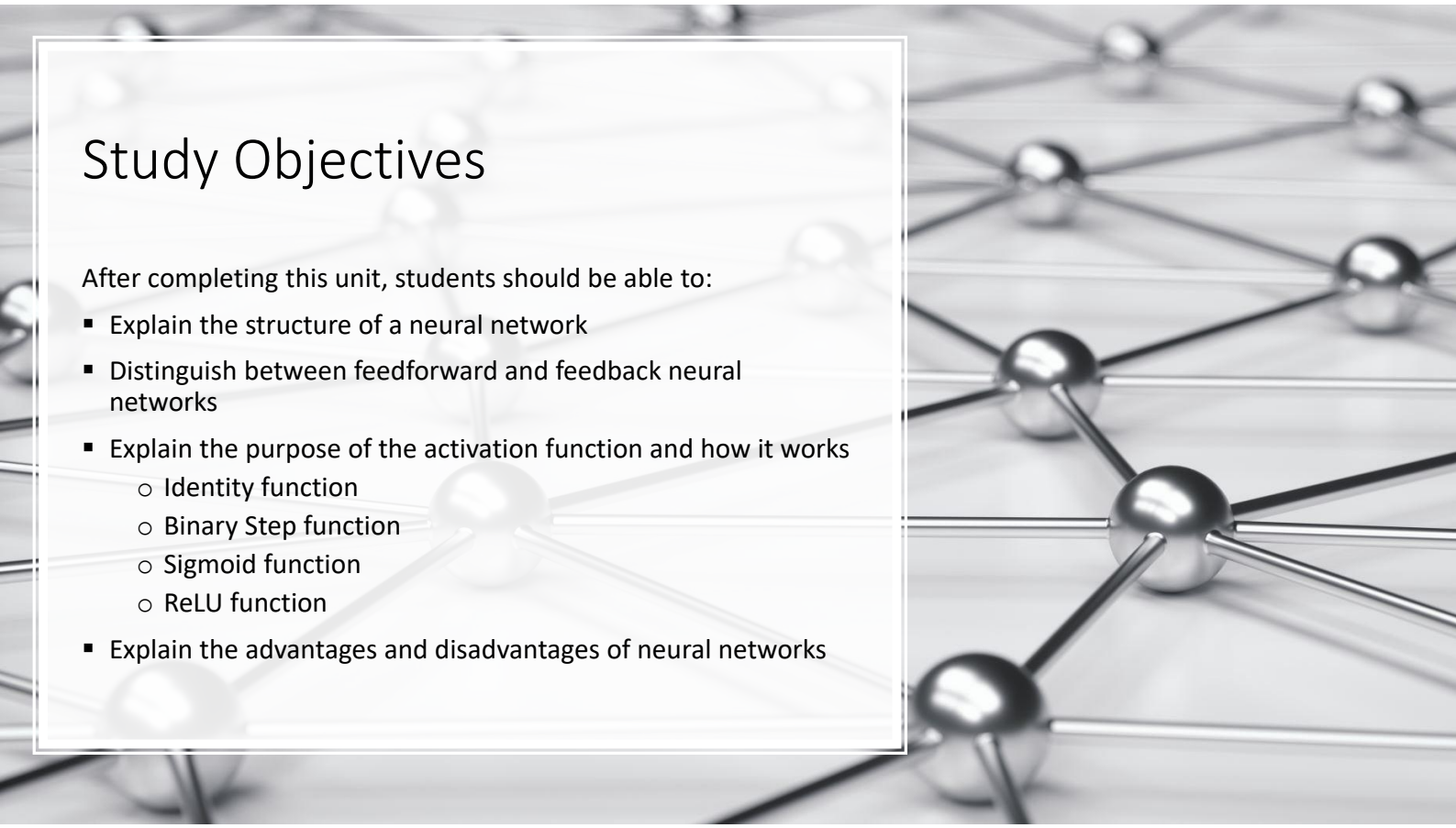
SVM……final thoughts

- SVMs are based on the idea of finding a hyperplane that best divides a dataset into classes.
- SVMs can handle data instances that are not immediately linearly seperable
- SVMs are a kernel-based classification approach where the kernels are represented in terms of a (possibly very large) subset of the training examples.
- SVMs uses the Kernel Trick to lift a non-linearly separable problem into a space where the data is linearly separable (or as near to separable as possible).
- SVMs are useful in cases where the useful interactions or other combinations of input variables aren't known in advance.



Unit 8- Neural Networks

Nana Boateng



Study Objectives

After completing this unit, students should be able to:

- Explain the structure of a neural network
- Distinguish between feedforward and feedback neural networks
- Explain the purpose of the activation function and how it works
  - Identity function
  - Binary Step function
  - Sigmoid function
  - ReLU function
- Explain the advantages and disadvantages of neural networks

Use Cases of Neural Networks

Pattern Recognition: neural networks are very suitable for pattern recognition problems such as facial recognition, object detection, fingerprint recognition, etc.

Anomaly Detection: neural networks are good at pattern detection, and they can easily detect the unusual patterns that don't fit in the general patterns.

Time Series Prediction: Neural networks can be used to predict time series problems such as stock price, weather forecasting.

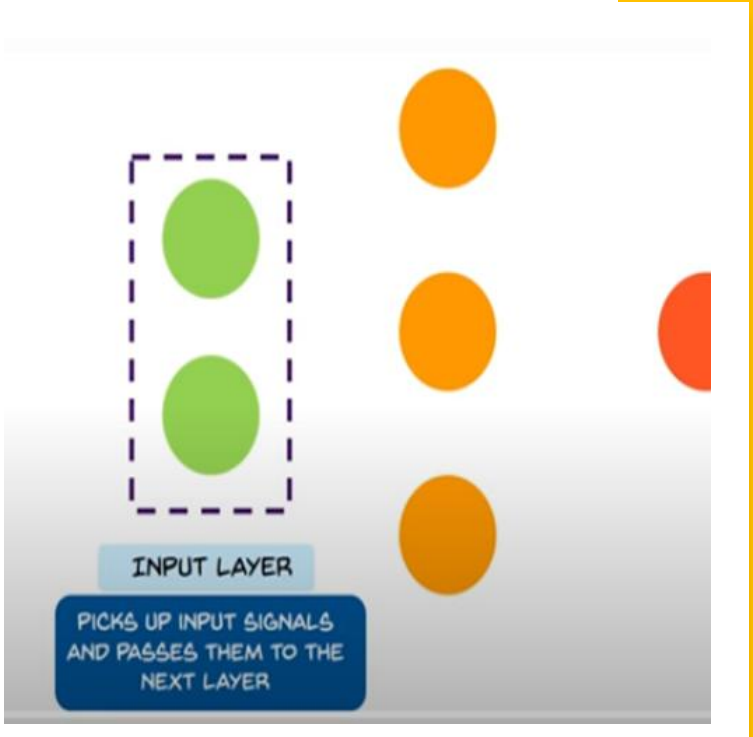
Natural Language Processing: Neural networks offer a wide range of applications in Natural Language Processing tasks such as text classification, Named Entity Recognition (NER), Part-of-Speech Tagging, Speech Recognition, and Spell Checking.

Neural Networks

- Neural Networks, also called Artificial Neural Networks (ANN), are models for classification and prediction.
- The neural network is based on a model of biological activity in the brain, where neurons are interconnected and learn from experience.
- The learning and memory properties of neural networks resemble the properties of human learning and memory, and they also have a capacity to generalise from particulars.

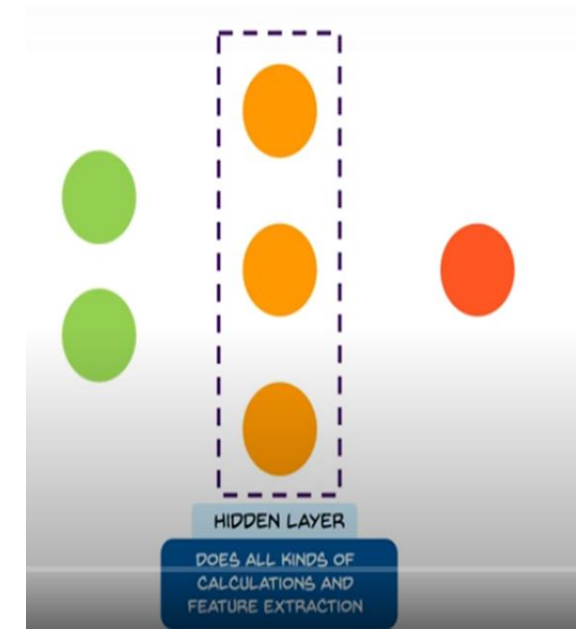
Structure of Neural Networks

- **Input Layer**
- Input layer consist of nodes (sometimes called neurons) that simply accept the predictor values, and successive layers of nodes that receive input from the previous layers.



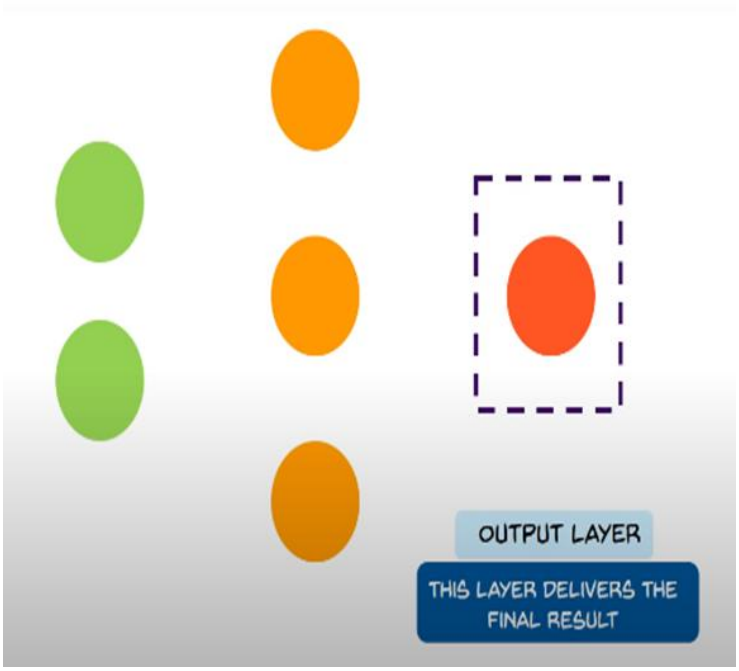
Structure of Neural Networks

- **Hidden Layer**
- Layers between the input and output layers are known as hidden layers.

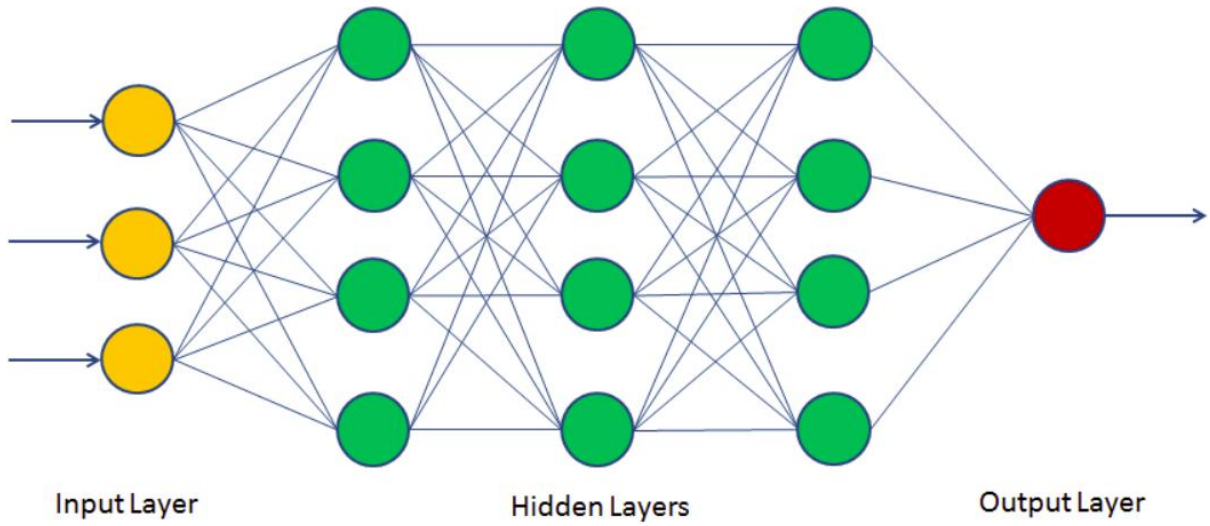


Structure of Neural Networks

- **Output Layer**
- The outputs of nodes in each layer are inputs to nodes in the next layer. The last layer is called the output layer.



Structure of Neural Networks



Feedforward & Feedback Artificial Neural Network

- There are two main types of artificial neural networks:
  - Feedforward artificial neural networks
  - Feedback artificial neural networks.

Feedforward Artificial Neural Networks

- Feedforward neural network is a network which is not recursive. Neurons in this layer were only connected to neurons in the next layer, and they are don't form a cycle.

- In Feedforward signals travel in only one direction towards the output layer.

Feedback Artificial Neural Networks

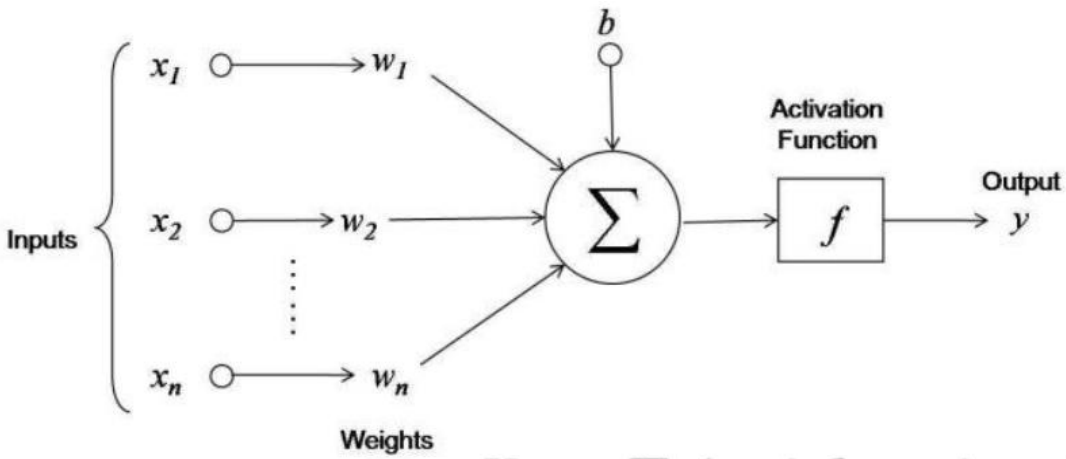
- Feedback neural networks contain cycles. Signals travel in both directions by introducing loops in the network.

- The feedback cycles can cause the network's behavior change over time based on its input. Feedback neural network also known as recurrent neural networks.

How the Neural Network Works

- The x1,x2....xn are input variables. w1,w2....wn are weights of respective inputs.
- b is the bias, which is summed with the weighted inputs to form the net inputs. Bias and weights are both adjustable parameters of the neuron.
- Parameters are adjusted using some learning rules. The output of a neuron can range from -inf to +inf. The neuron does not know the boundary.
- So, we need a mapping mechanism between the input and output of the neuron. This mechanism of mapping inputs to output is known as Activation Function.

How the Neural Network Works



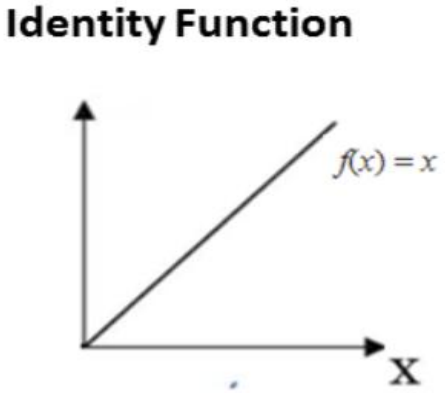
$$Y = \sum (weight * input) + bias$$

Activation Functions

- Activation function defines the output of a neuron in terms of a local induced field. Activation functions are a single line of code that gives the neural nets non-linearity and expressiveness.
- Examples of activation functions include:
  - Identity functions
  - Binary Step Function
  - Sigmoid Function
  - Binary Sigmoid Function

Activation Functions - Identity Function

- Identity function is a function that maps input to the same output value. It is a linear operator in vector space.
- Also, known straight line function where activation is proportional to the input.

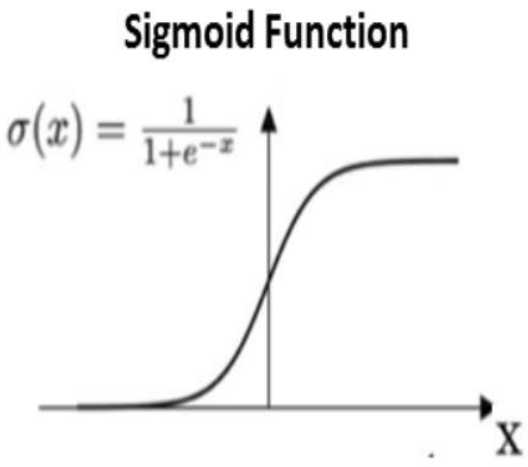


Activation Functions - Binary Step Function

- In Binary Step Function, if the value of Y is above a certain value known as the threshold, the output is True(or activated), and if it's less than the threshold, then the output is false (or not activated). It is very useful in the classifier.

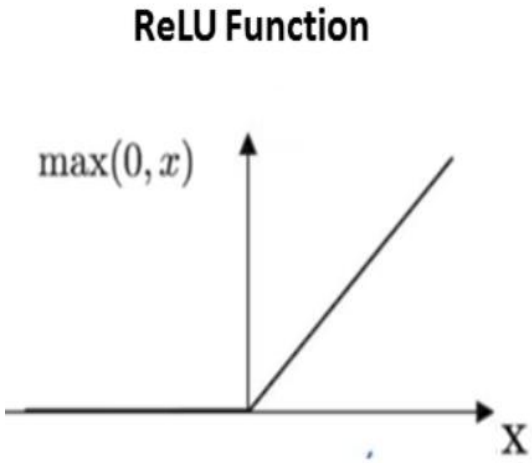
Activation Functions - Sigmoid Function

- **Sigmoid Function** called S-shaped functions. Logistic and hyperbolic tangent functions are commonly used sigmoid functions.
- There are two types of sigmoid functions.
  - **Binary Sigmoid Function** is a logistic function where the output values are either binary or vary from 0 to 1.
  - **Bipolar Sigmoid Function** is a logistic function where the output value varies from -1 to 1. Also known as Hyperbolic Tangent Function or tanh.



Activation Functions - ReLU Function

- **ReLu** stands for the rectified linear unit (ReLU). It is the most popular activation function.
- It output 0 for negative values of x.



Adavantages of Neural Networks

- Neural networks are more flexible and can be used with both regression and classification problems.
- Neural networks are good for the nonlinear dataset with a large number of inputs such as images.
- Neural networks can work with any number of inputs and layers.
- Neural networks have the numerical strength that can perform jobs in parallel.

Disadvantages of Neural Networks

- There are more alternative algorithms such as SVM, Decision Tree and Regression are available that are simple, fast, easy to train, and provide better performance.
- Neural networks are much more of the black box, require more time for development and more computation power.
- Neural Networks requires more data than other Machine Learning algorithms. NNs can be used only with numerical inputs and non-missing value datasets.