Chapter 4

Detailed Design of classification of unstructured data from online course web pages

Detailed Design is a phase where in the internal logic of each of the modules specified in high-level design is specified. In this phase further details and algorithmic design of each of the modules is specified. Other low-level components and sub-components are also described as well. Each subsection of this section will refer to or contain a detailed description of system software component. This chapter also discusses about the control flow in the software with much more details about software modules by clarifying the details about each function with functionality, purpose, input, and output. This chapter presents the following:

- Structure Chart for the project
- Functional Description of units
- Flowchart for units

4.1 Structure Chart

A Structure Chart shows the control flow among the units in a system. The Structure Chart explains the identified units and the interaction between the units. It also explains the sub-modules. It explains the input for each unit and output generated.

There are 5 main modules to this project:

1. Crawler

2. Featurizer

3. Classifier

4. Database

5. User Interface

The sequence of flow and control of each module/sub-module is shown in figure 4.1.

This is a structured chart that represents all the modules and the data that flows in them.

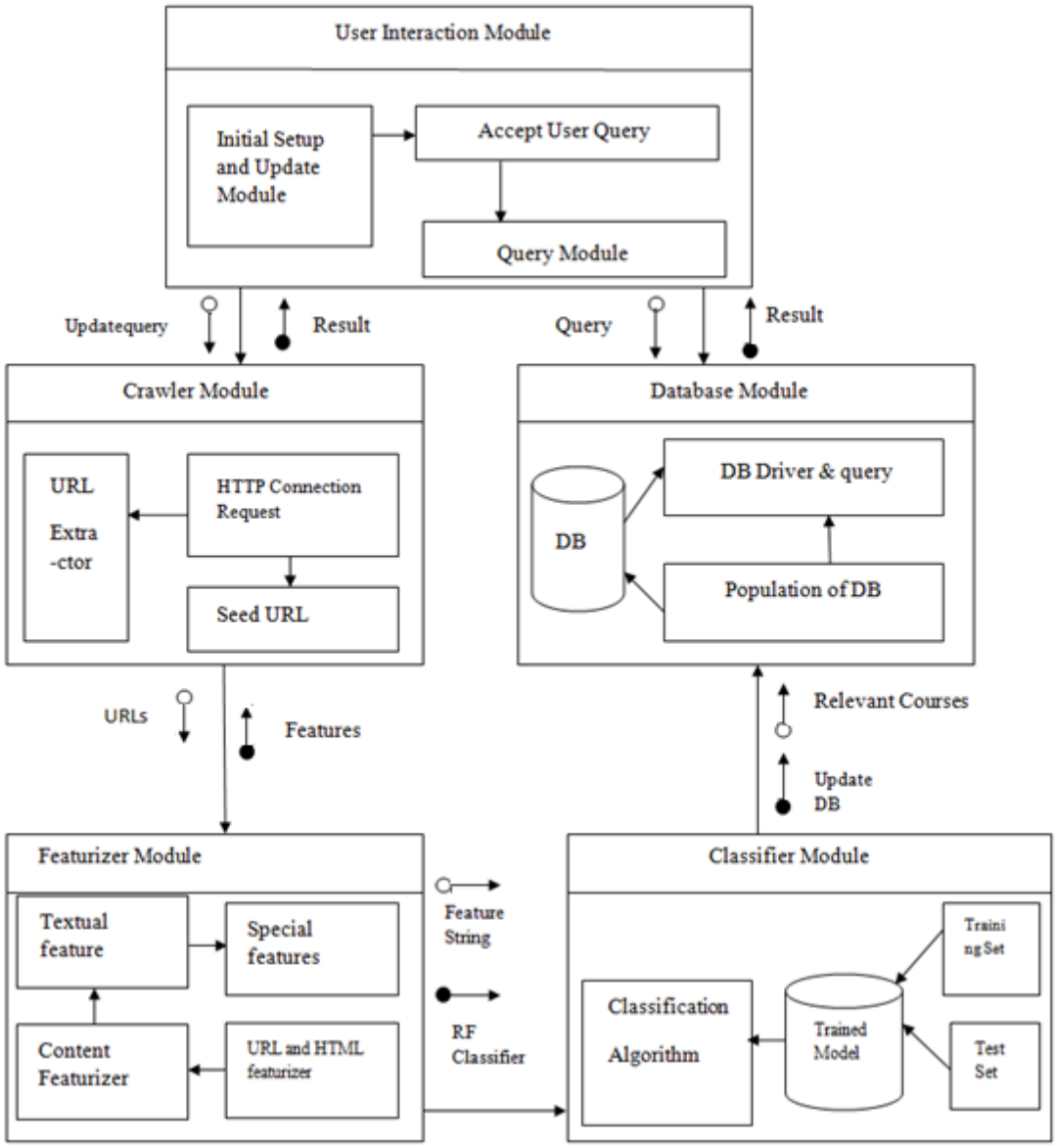

Figure 4.1: Structure chart of classification of unstructured data from online course web pages

4.2 Functional Description of Modules

This section contains a detailed description of software components, low-level components and other sub components of the project.

4.2.1 Crawler module

This section contains some information on the functionality and some software component attributes of the Crawler Module. Figure 4.2 is the flow chart for the crawler module. The crawler is first initialized with seed urls. After the initialization http request

is sent to download the page. The feature module is then initialized if the the web page is of online course.

- **Purpose:** The purpose of this module is to crawl the online courses related web urls.
- **Functionality:** The functionality of this module is to use a multi-threaded crawler to crawl the web for urls upto the depth of 5 to ensure all the related web pages are covered.
- **Input:** Intialize the crawler with seed urls.
- Output: The output is the list of urls. These urls are fed into the featurizer as an input.
- **Flowchart:** The flowchart for the given module is given as follows in figure 4.2. This is the flowchart for the Crawler module. This flowchart indicates how the crawler performs the function of scanning the web and fetching pages. This is done within a particular domain beginning from an initial seed variable specified and up to the configured depth of crawling.

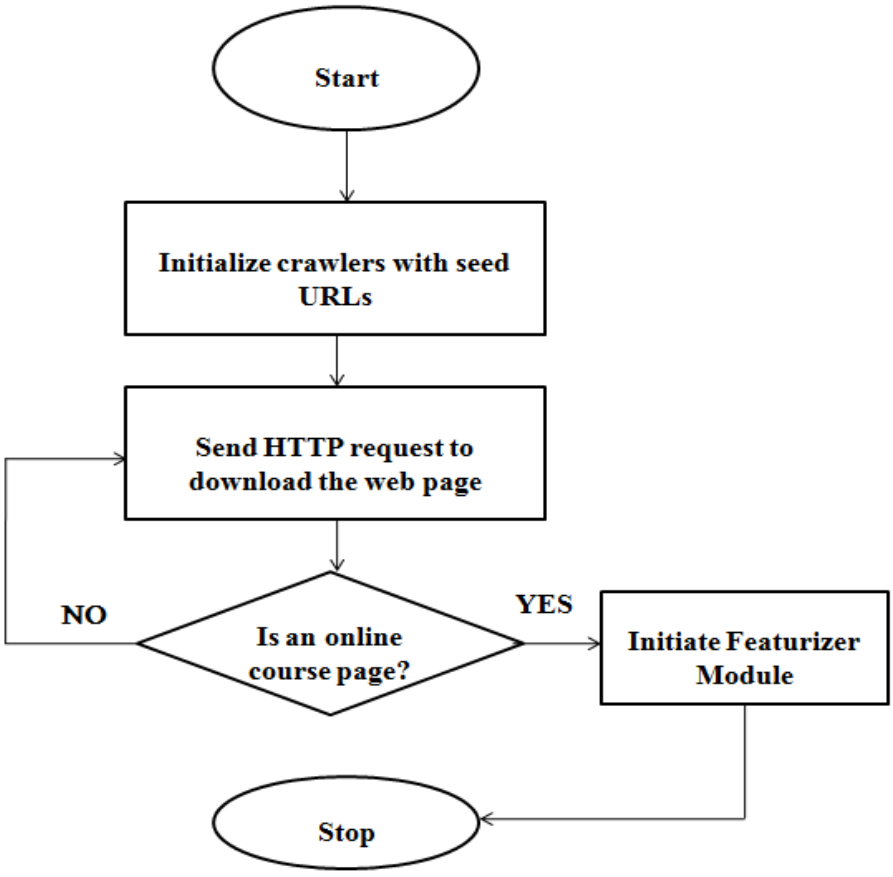

Figure 4.2: Flow chart for crawler

4.2.2 Featurizer module

This section contains some information on the functionality and some software component attributes of the Featurizer Module. Figure 4.3 is the flow chart for the featurizer module.

- **Purpose**: The purpose of this module is to extract features from the web pages crawled urls of the online course webpages.
- **Functionality**: The functionality of this module is to extract features such as the title of the webpage, meta-description, course name, keywords, social tags, provider, price, duration, certificate, instructor, description and level.
- **Input**: The urls crawled by the crawler.
- **Output**: The output will be the feature string that essentially contains url features,content features,social featurs,HTML features. This module structures the data from the web pages.
- **Flowchart**: The flowchart for the given module is given as follows in figure 4.3.

First step of the module is to obtain the list of urls crawled. The features are then extracted such as urls features, content features, social features, HTML features. The set of features extracted are title of the web page, meta-description, course name, keywords, social tags, provider, price, duration, certificate, instructor, description and level. These features are extracted based on the type of domain as well as the specific pages within the domain. Various scripting language and regular expression based methods are used to extract the required features and they are concatenated into a single comma separated string so that they can directly be given to the classifier and further modules in the system.

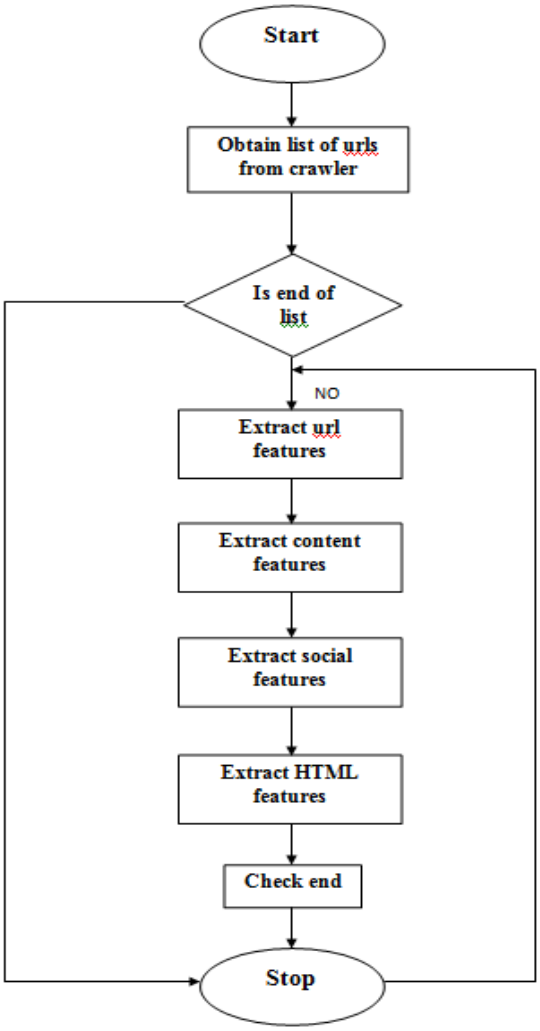

Figure 4.3: Flow chart for featurizer

4.2.3 Classifier module

This section contains some information on the functionality and some software component attributes of the classifier Module

- **Purpose:** The purpose of this module is to classify the featurized web pages as relevant or irrelevant pages.
- **Functionality:** A model is trained initially with a trained dataset. It is bootstrapped with a trained set. The trained model is then applied on the the test dataset to classify the rest of the web pages. Classification algorithms are applied using the trained model for classification.
- **Input:** The feature string which was the output of the featurizer module.

- **Output:** Classified data indicating the relevance for each of the web page that was crawled. This data is then stored in the database.
- **Flowchart:** The flowchart for the given module is given as follows in figure 4.4.

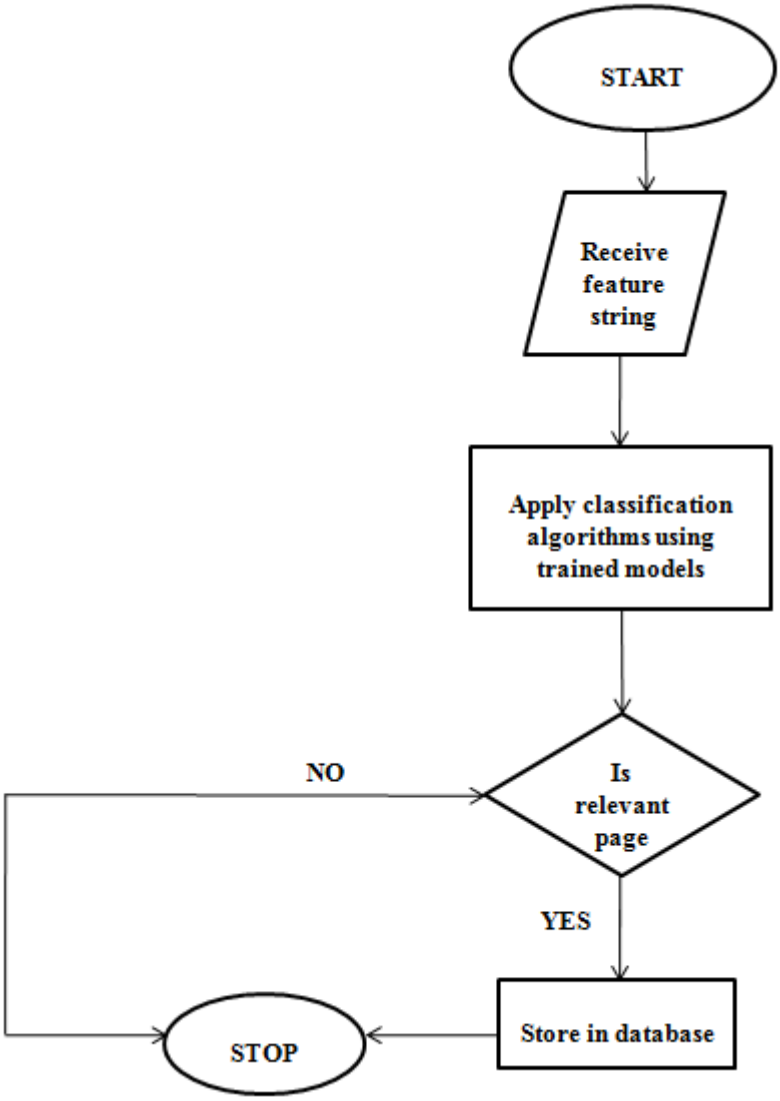

Figure 4.4: Flow chart for classifier

The first step of the module is to receive feature string obtained from the featurizer. Classification algorithms are then applied upon the recieved feature string using trained models. The classifier then classifies the web page into relevant online course web page. The relevant online course pages are then stored in the database.

4.2.4 Database module

This section contains some information on the functionality and some software component attributes of the database Module

- **Purpose:** The purpose of this module is store the classified structured web page data.
- **Functionality:** The functionality of this module is to retrieve relevant results for user query and return the results to the user.
- **Input:** The user query.The user query is made in the form of an SQL statement by the module.
- **Output:** The output will be the results for the query made.
- **Flowchart:** The flowchart for the given module is given as follows in figure 4.5.

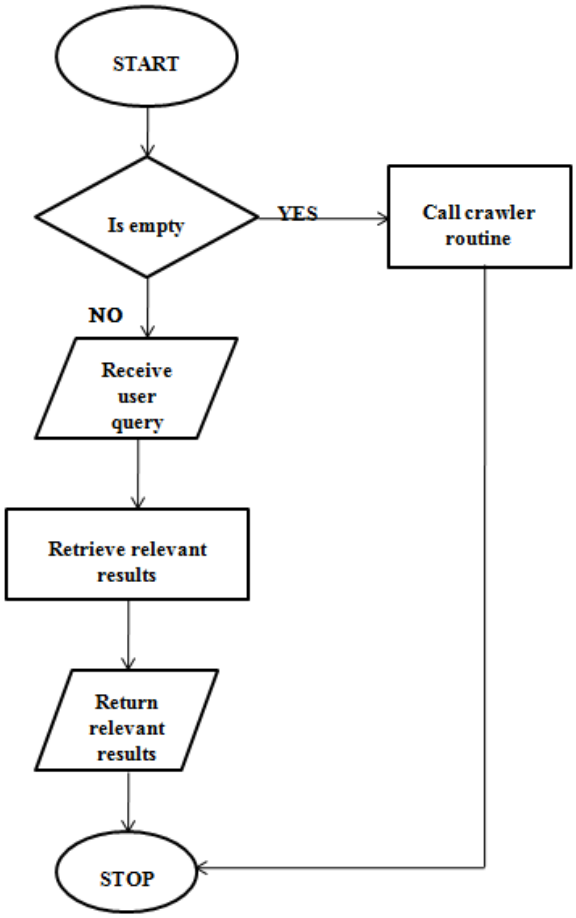

Figure 4.5: Flow chart for database module

The first step begins with received user query. Relevant results i.e the relevant online course web pages are returned that satisfies the user query. If the database is empty crawler routine is called.

4.2.5 User Interface module

This section contains some information on the functionality and some software component attributes of the user interface module

- **Purpose:** The purpose of this module is to design a user friendly front end.
- **Functionality:** The functionality of this module to accept the query requests from the user and retrieve the results from the database. The main functionality is to display the information in a consolidated and a structured manner. It also contains an update module.The update module delivers the data about the new courses introduced in the recent past. It calls the crawler routine to crawl the web for new course and delivers an upto date information to the users.
- **Input:** The user query.
- **Output:**Results for the query made by the user.
- **Flowchart:** The flowchart for the given module is given as follows in figure 4.6.

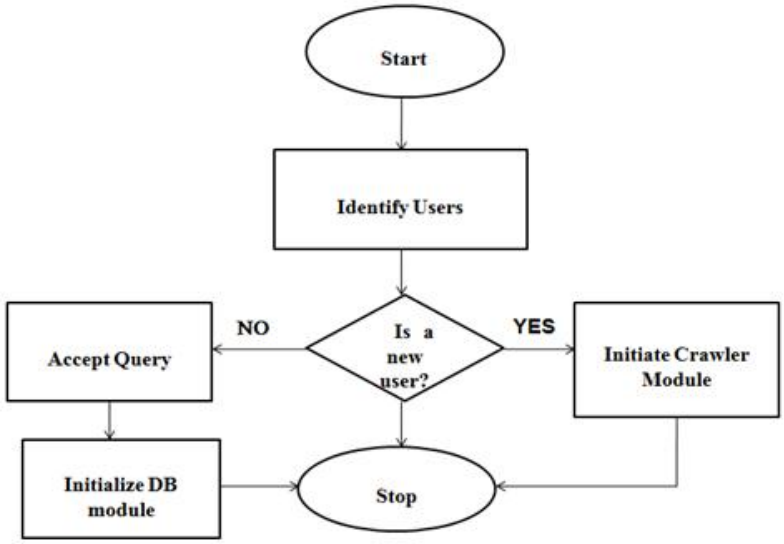

Figure 4.6: Flow chart for user interface module

4.3 Algorithms

Five classification algorithms were chosen for comparing the accuracy in labelling the online course web pages as relevant courses. The following are the algorithms compared which are discussed in detail.

1. C4.5
2. Logistic Regression
3. Naive bayes
4. K Nearest Neighbor
5. Random Forest

4.3.1 C4.5 Algorithm

C4.5 is an algorithm used to generate a decision tree. C4.5 builds decision trees from a set of training data. The training data is a set $S = s_1,s_2,s_{3.....}$ of already classified samples. Each sample S_i consists of a p-dimensional vector $(x_{1,i},x_{2,i},.....,x_{p,i})$, where the x_j represent attributes or features of the sample, as well as the class in which S_i falls. At each node of the tree, C4.5 chooses the attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other. The splitting criterion is the normalized information gain (difference in entropy). The attribute with the highest normalized information gain is chosen to make the decision. The C4.5 algorithm then recurses on the smaller sublists [44].

This algorithm has a few base cases.

- All the samples in the list belong to the same class. When this happens, it simply creates a leaf node for the decision tree saying to choose that class.
- None of the features provide any information gain. In this case, C4.5 creates a decision node higher up the tree using the expected value of the class.
- Instance of previously-unseen class encountered. Again, C4.5 creates a decision node higher up the tree using the expected value.

The general algorithm for building decision trees is:

1. Check for base cases for each attribute a Find the normalized information gain ratio from splitting on *a*

2. Let *a_best* be the attribute with the highest normalized information gain
3. Create a decision *node* that splits on *a_best*
4. Recurse on the sublists obtained by splitting on *a_best*, and add those nodes as children of *node.*

4.3.2 Logistic Regression algorithm

Logistic regression or logit regression is a type of probabilistic statistical classification model. It is also used to predict a binary response from a binary predictor, used for predicting the outcome of a categorical dependent variable (i.e., a class label) based on one or more predictor variables (features). That is, it is used in estimating empirical values of the parameters in a qualitative response model. The probabilities describing the possible outcomes of a single trial are modeled, as a function of the explanatory (predictor) variables, using a logistic function. Frequently (and subsequently in this article) "logistic regression" is used to refer specifically to the problem in which the dependent variable is binary—that is, the number of available categories is two—and problems with more than two categories are referred to as multinomial logistic regression or, if the multiple categories are ordered, as ordered logistic regression.

Logistic regression measures the relationship between a categorical dependent variable and one or more independent variables, which are usually (but not necessarily) continuous, by using probability scores as the predicted values of the dependent variable.As such it treats the same set of problems as does probit regression using similar techniques [45].

An explanation of logistic regression begins with an explanation of the logistic function, which always takes on values between zero and one as shown in equation (1):

$$F(t) = \frac{e^t}{e^t+1} = \frac{e^t}{e^{-t}+1} \quad (1)$$

viewing *t* as a linear function of an explanatory variable x (or of a linear combination of explanatory variables), the logistic function can be written as in equation (2) given below:

$$F(x) = \frac{1}{1+ e^{-(\beta_0+\beta_1 x)}} \quad (2)$$

This will be interpreted as the probability of the dependent variable equalling a "success" or "case" rather than a failure or non-case. We also define the inverse of the logistic function, the logit as in equation (3):

$$g(x) = ln\frac{F(x)}{1-F(x)} = \beta_0+\beta_1 x \quad (3)$$

and equivalently in equation 4:

$$\frac{F(x)}{1-F(x)} = e^{\beta_0+\beta_1 x} \quad (4)$$

A graph of the logistic function *F(x)* is shown in Figure 4.6. The input is the value of $\beta_0+\beta_1 x$ and the output is *F(x)*. The logistic function is useful because it can take an input with any value from negative infinity to positive infinity, whereas the output *F(x)* is confined to values between 0 and 1 and hence is interpretable as a probability. In the above equations, *g(x)* refers to the logit function of some given linear combination x of the predictors, $\ln$ denotes the natural logarithm, *F(x)* is the probability that the dependent variable equals a case, β_0 is the intercept from the linear regression equation (the value of the criterion when the predictor is equal to zero), $\beta_1 x$ is the regression coefficient multiplied by some value of the predictor, and base e denotes the exponential function.

The formula for *F(x)* illustrates that the probability of the dependent variable equaling a case is equal to the value of the logistic function of the linear regression expression. This is important in that it shows that the value of the linear regression expression can vary from negative to positive infinity and yet, after transformation, the resulting expression for the probability *F(x)* ranges between 0 and 1. The equation for *g(x)* illustrates that the logit (i.e., log-odds or natural logarithm of the odds) is equivalent to the linear regression expression. Likewise, the next equation illustrates that the odds of the dependent variable equaling a case is equivalent to the exponential function of the linear regression expression. This illustrates how the logit serves as a link function between the probability and the linear regression expression. Given that the logit ranges between minus infinity and infinity, it provides an adequate criterion upon which to conduct linear regression and the logit is easily converted back into the odds.

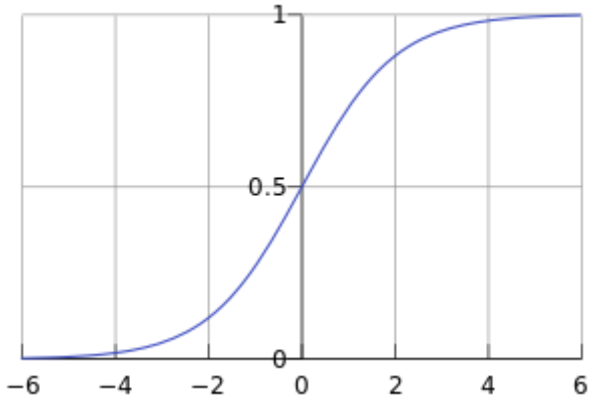

Figure 4.6. The logistic function, with $\beta_0+\beta_1 x$ horizontal axis and *F(x)* on the vertical axis

4.3.3 Naive Bayes algorithm

A naive Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem with strong (naive) independence assumptions. A more descriptive term for the underlying probability model would be "independent feature model" [46].

The Bayes Naive classifier selects the most likely classification V_{nb} given the attribute values $a_1,a_2,a_3...a_n$.

This results in:

$$V_{nb} = \text{argmax}_{v_j \in V} P(v_j) \prod P(a_i|v_j) \quad (5)$$

Where $P(a_i|v_j)$ is estimated by:

$$P(a_i|v_j) = \frac{nc + mp}{n+m} \quad (6)$$

where:

n = the number of training examples for which $v = v_j$

n_c = number of examples for which $v = v_j$ and $a = a_i$

p = a priori estimate for P(ai|vj)

m = the equivalent sample size

4.3.4 K Nearest Neighbor algorithm

k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms.

KNN is a method for classifying objects based on closest training examples in the feature space. An object is classified by a majority vote of its neighbors. K is always a positive integer. The neighbors are taken from a set of objects for which the correct classification is known. It is usual to use the Euclidean distance, though other distance measures such as the Manhattean distance could in principle be used instead [47]. The algorithm on how to compute the K-nearest neighbors is as follows:

1. Determine the parameter K = number of nearest neighbors before hand. This value is all up to you.
2. Calculate the distance between the query-instance and all the training samples. You can use any distance algorithm.
3. Sort the distances for all the training samples and determine the nearest neighbor based on the K-th minimum distance.
4. Since this is supervised learning, get all the Categories of your training data for the sorted value which fall under K.
5. Use the majority of nearest neighbors as the prediction value.

4.3.5 Random Forest algorithm

Random forests are an ensemble learning method for classification (and regression) that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes output by individual trees.

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set X = 1, …, xn with responses Y= y1 through yn, bagging repeatedly selects a bootstrap sample of the training set and fits trees to these samples:

For b = 1 through B:

1. Sample, with replacement, n training examples from X, Y; call these X_b, Y_b.
2. Train a decision or regression tree f_b on X_b, Y_b.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x' as shown in equation (7):

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} \hat{f}(x') \quad (7)$$

or by taking the majority vote in the case of decision trees.

In the above algorithm, B is a free parameter. Typically, a few hundred to several thousand trees are used, depending on the size and nature of the training set. Increasing the number of trees tends to decrease the variance of the model, without increasing the bias. As a result, the training and test error tend to level off after some number of trees has been fit. An optimal number of trees B can be found using cross-validation, or by observing the out-of-bag error: the mean prediction error on each training sample x_i, using only the trees that did not have x_i in their bootstrap sample [48].

4.4 Summary

This chapter gives the detail layout of the architecture. It discusses all the modules in detail and explains its working. The detail explanation includes the purpose, functionality, input, output and flow chart of each module. Also all the algorithms used for comparison are explained in this chapter. The comparison and results of the algorithms are discussed in later chapters.