Chapter 1

Introduction

A course recommender system is one of the most powerful tools a student can get his hands on. It is a helpful tool in terms of being effective and providing suggestions to the student. Course Recommendation System is necessary as many e-learning courses are available online. In order to choose the right course for their study is very crucial. The proposed system would be beneficial for the students looking to gain knowledge in an optimized manner. A student can learn new things from multiple websites like Udemy, Coursera etc. where one can search the course they desire. A very basic issue with the students is to find the BEST course for their study.

Ratings can be deceiving. The student community faces the issue of finding the most suitable course for their study. Keeping this in mind, an online course recommendation system is designed to help the students at large. MOOC_RecSys is designed to understand, analyze the needs of a student (primarily of technical background). Since this is a student centric approach, so in this project, the user and student are used analogously. The course recommender system would suggest the best suitable courses for the students after making a knowledge base and finding a similar user to them. This recommendation is based on a personal evaluation of the user based on some predefined attributes. The system is designed to be able to do a semantic analysis of the available online user's reviews [4] and thus provides useful recommendations. MOOC_RecSys considers a set of courses in different areas as the Domain of Recommendation (DoR). In this project, for simplicity, the DoR is taken as Computer Science and Engineering. From the DoR, once a class of similar users is found, their recommendation are shown user.

This approach is unique as for considering the similar users the feature set [4] contains the user attributes along with course reviews given by the existing users. The reviews help in determining if a user liked or disliked the course. Once the review is analyzed using the proposed sentiment analysis algorithm, a rating indicating this like/dislike factor for the

existing user can be determined. The courses for which the similar users give a like rating will be chosen for the recommendation. The proposed system should have a User Interface for the new user, where he or she can enter his or her details and search for the course needed and should also contain a User Interface which would show the courses recommended. The proposed system has difficulty in finding similar users as the existing user's data needs to be updated, and the clusters or the classes have to be measured with less abstraction than the current implementation.

1.1 Recommendation System

A recommender system or a recommendation system is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item. Recommender systems are a useful alternative to search algorithms since they help users discover items they might not have found otherwise. Of note, recommender systems are often implemented using search engines indexing non-traditional data. Recommender systems typically produce a list of recommendations in one of two ways – through collaborative filtering or through content-based filtering. Collaborative filtering approaches build a model from a user's past behaviour and as well as similar decisions made by other users. This model is then used to predict items or ratings for items that the user may have an interest in. Content-based filtering approaches utilize a series of discrete characteristics of an item in order to recommend additional items with similar properties and these approaches are often combined by Hybrid Recommender Systems.

Recommender systems have become increasingly popular in recent years, and are utilized in a variety of areas including movies, music, news, books, research articles, search queries, social tags, and products in general. Mostly used in the digital domain, majority of today's E-Commerce sites like eBay, Amazon, Alibaba etc make use of their proprietary recommendation algorithms in order to serve better the customers with the products they are bound to like.

1.2 Types Of Recommendation

There are mainly three types of recommendation systems being used as discussed below.

1.2.1 Collaborative Filtering

Collaborative filtering, also referred to as social filtering, filters information by using the recommendations of other people. It is based on the idea that people who agreed in their evaluation of certain items in the past are likely to agree again in the future. A person who wants to see a movie for example, might ask for recommendations from friends. The recommendations of some friends who have similar interests are trusted more than recommendations from others. This information is used in the decision on which movie to see. This filtering method is usually based on collecting and analyzing information on user's behaviors, their activities or preferences and predicting what they will like based on the similarity with other users (as seen in the figure 1.1). A key advantage of the collaborative filtering approach is that it does not rely on machine analyzable content and thus it is capable of accurately recommending complex items such as movies without requiring an "understanding" of the item itself. Collaborative filtering is based on the assumption that people who agreed in the past will agree in the future, and that they will like similar kinds of items as they liked in the past. For example, if a person A likes item 1, 2, 3 and B like 2, 3, 4 then they have similar interests and A should like item 4 and B should like item 1.

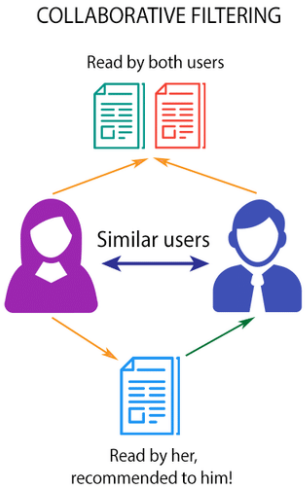

Figure 1.1: Collaborative based filtering system

1.2.2 Content Based Recommendation System

These filtering methods are based on the description of an item and a profile of the user's preferred choices. In a content-based recommendation system, keywords are used to describe the items; besides, a user profile is built to state the type of item this user likes. (As seen the figure 1.2) , the algorithms try to recommend products which are similar to the ones that a user has liked in the past. The idea of content-based filtering is that if you like an item you will also like a 'similar' item for example, when we are recommending the same kind of item like a movie or song recommendation. This approach has its roots in information retrieval and information filtering research.

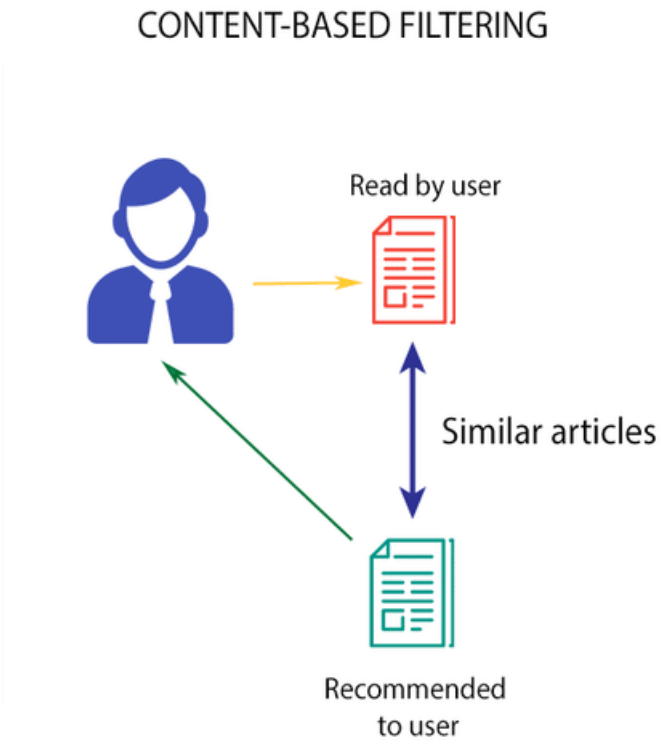

Figure 1.2: Content based filtering system

1.2.3 Hybrid Recommendation System

Recent research shows that combining collaborative and content-based recommendation can be more effective. Hybrid approaches can be implemented by making content-based and collaborative-based predictions separately and then combining them. Further, by adding content-based capabilities to a collaborative-based approach and vice versa; or by unifying the approaches into one model (as seen in the figure 1.3). Several studies focused on comparing the performance of the hybrid with the pure collaborative and content-based methods and demonstrate that hybrid methods can provide more accurate recommendations than pure approaches. Such methods can be used to overcome the common problems in recommendation systems such as cold start and the data paucity problem. Netflix is a good example, which uses hybrid recommender systems to recommend daily shows. The website makes recommendations by comparing the watching and searching habits of similar users (i.e., collaborative filtering) as well as by offering movies that share characteristics with films that a user has rated highly (content-based filtering).

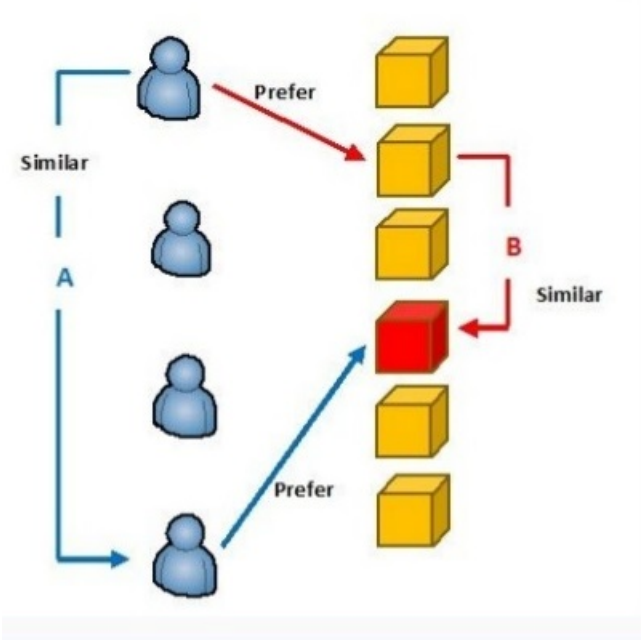

Figure 1.3: Hybrid based recommendation system

1.3 Need For Project

The proposed project suggests an optimum course for each student separately. This gives each user the flexibility to try new course which they will be able to complete within stipulated time and gain knowledge without having to worry if the suggested course would neither be too easy nor too difficult for them. This system also saves the time of user which could have been lost due to sheer number of online courses available on the internet.

1.4 Problem Statement

To get the data from various data sources such as Kaggle, UCL etc. To analyze the reviews using sentiment analysis which involves generating a classifier and testing it. To recommend online courses to users according to the courses suggested to the similar users before. It is quite a difficult task for a student to be able to take wise decisions about the courses he has to take up specially, if he is a new student and has few contacts to refer to. Even if he did have the contacts it would be subjective to a few opinions, which might not ensure the quality of the decision. Moreover this process could result in students choosing courses that might not be totally suited to their liking as it is hard for them to assess the course based on a few opinions. Taking a course without much value to the student would mean monetary wastage as well as wastage of time both of which are very crucial resources to a student.

Chapter 2

Related Work

Recommendations in the field of education have been a phenomenon in the last decade. The volume of e-learning content has grown exponentially and so has the parallel research for efficiently learning new concepts and subjects. In the last years, a number of works have focused on the use of data mining and machine learning techniques in the context of the educational environment for instance. Educationally Adaptive HyperMedia (EAH)[9]. These techniques are used by online courses to discover patterns which can be helpful in making the courses better for professors and students. In the literature, research has also been done on recommendation models based on the feedback data gathered from student's assessment of courses [9] relevant to their career goals. Along with this many authors have also researched in the field of encouraging users' contribution [10] in getting a recommendation for the choice they might take. Many authors have also analyzed the application of data mining techniques to the recommender System. Some of which have been dealing with a reward based system so as to increase a user's contribution[10].Others have been based on collaborative and hybrid techniques of classification.

In the proposed MOOC_RecSys, the task of finding similar users is computed using cosine similarity distance and applying the hierarchal clustering model [5]. This approach has shown greater accuracy than a knowledge based approach where the mean of values are taken without considering the context of courses the users have done earlier. Thus, to understand the collaboration of desired courses with users (and subsequent similar users) motivates to perform sentiment analysis of online reviews of various courses and hence the proposed approach is specific for the collaborative recommender system use-case.

2.1 Machine Learning

Recommender systems has one progressive step in its history which is the adoption of machine learning (ML) algorithms, which allow computers to learn based on user information and to personalize recommendations further. Machine learning is an Artificial

Intelligence (AI) research field that encompasses algorithms whose goal is to predict the outcome of data processing. Machine learning has a variety of algorithms that can be used in the recommendation system like Naïve bayes, Decision tree, matrix-factorisation based, neural networks, neighbor - based and many more. In Recommendation system development, the ML algorithm used is a Bayesian or decision tree approach in most of the cases. Their recent popularity and the low complexity in calculation and implementation contribute to this result. Moreover, most of the approaches have a mathematical or statistical description of algorithms, since the ML field has origins in mathematics and statistics.

2.2 Natural Language Processing

Natural language processing (NLP) is a subfield of computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data.

The history of natural language processing generally started in the 1950s, although work can be found from earlier periods. In 1950, Alan Turing published an article titled "Intelligence" which proposed what is now called the Turing test as a criterion of intelligence. The Georgetown experiment in 1954 involved fully automatic translation of more than sixty Russian sentences into English. The authors claimed that within three or five years, machine translation would be a solved problem. However, real progress was much slower, and after the ALPAC report in 1966, which found that ten-year-long research had failed to fulfill the expectations, funding for machine translation was dramatically reduced. Little further research in machine translation was conducted until the late 1980s, when the first statistical machine translation systems were developed.

Some notably successful natural language processing systems developed in the 1960s were SHRDLU, a natural language system working in restricted "blocks worlds" with restricted vocabularies, and ELIZA, a simulation of a Rogerian psychotherapist, written by Joseph Weizenbaum between 1964 and 1966. Using almost no information about human thought or

emotion, ELIZA sometimes provided a startlingly human-like interaction. When the "patient" exceeded the very small knowledge base, ELIZA might provide a generic response, for example, responding to "My head hurts" with "Why do you say your head hurts?".

During the 1970s, many programmers began to write "conceptual ontologies", which structured real-world information into computer-understandable data. Examples are MARGIE (Schank, 1975), SAM (Cullingford, 1978), PAM (Wilensky, 1978), TaleSpin (Meehan, 1976), QUALM (Lehnert, 1977), Politics (Carbonell, 1979), and Plot Units (Lehnert 1981). During this time, many chatterbots were written including PARRY, Racter, and Jabberwacky.

Up to the 1980s, most natural language processing systems were based on complex sets of hand-written rules. Starting in the late 1980s, however, there was a revolution in natural language processing with the introduction of machine learning algorithms for language processing. This was due to both the steady increase in computational power (see Moore's law) and the gradual lessening of the dominance of Chomskyan theories of linguistics (e.g. transformational grammar), whose theoretical underpinnings discouraged the sort of corpus linguistics that underlies the machine-learning approach to language processing. Some of the earliest-used machine learning algorithms, such as decision trees, produced systems of hard if-then rules similar to existing hand-written rules. However, part-of-speech tagging introduced the use of hidden Markov models to natural language processing, and increasingly, research has focused on statistical models, which make soft, probabilistic decisions based on attaching real-valued weights to the features making up the input data. The cache language models upon which many speech recognition systems now rely are examples of such statistical models. Such models are generally more robust when given unfamiliar input, especially input that contains errors (as is very common for real-world data), and produce more reliable results when integrated into a larger system comprising multiple subtasks.

Many of the notable early successes occurred in the field of machine translation, due especially to work at IBM Research, where successively more complicated statistical models

were developed. These systems were able to take advantage of existing multilingual textual corpora that had been produced by the Parliament of Canada and the European Union as a result of laws calling for the translation of all governmental proceedings into all official languages of the corresponding systems of government. However, most other systems depended on corpora specifically developed for the tasks implemented by these systems, which was (and often continues to be) a major limitation in the success of these systems. As a result, a great deal of research has gone into methods of more effectively learning from limited amounts of data. Recent research has increasingly focused on unsupervised and semi-supervised learning algorithms. Such algorithms are able to learn from data that has not been handannotated with the desired answers, or using a combination of annotated and non-annotated data. Generally, this task is much more difficult than supervised learning, and typically produces less accurate results for a given amount of input data. However, there is an enormous amount of non-annotated data available (including, among other things, the entire content of the World Wide Web), which can often make up for the inferior results if the algorithm used has a low enough time complexity to be practical.

In the early days, many language-processing systems were designed by hand-coding a set of rules, e.g. by writing grammars or devising heuristic rules for stemming. However, this is rarely robust to natural language variation.

Since the so-called "statistical revolution" in the late 1980s and mid 1990s, much natural language processing research has relied heavily on machine learning. The machine-learning paradigm calls instead for using statistical inference to automatically learn such rules through the analysis of large corpora of typical real-world examples (a corpus (plural, "corpora") is a set of documents, possibly with human or computer annotations).

Many different classes of machine-learning algorithms have been applied to natural language-processing tasks. These algorithms take as input a large set of "features" that are generated from the input data. Some of the earliest-used algorithms, such as decision trees, produced systems of hard if-then rules similar to the systems of hand-written rules that were then common. Increasingly, however, research has focused on statistical models, which make

soft, probabilistic decisions based on attaching real-valued weights to each input feature. Such models have the advantage that they can express the relative certainty of many different possible answers rather than only one, producing more reliable results when such a model is included as a component of a larger system.

2.3 K-means Algorithm

K-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

The problem is computationally difficult (NP-hard); however, there are efficient heuristic algorithms that are commonly employed and converge quickly to a local optimum. These are usually similar to the expectation-maximization algorithm for mixtures of Gaussian distributions via an iterative refinement approach employed by both k-means and Gaussian mixture modeling. Additionally, they both use cluster centers to model the data; however, k-means clustering tends to find clusters of comparable spatial extent, while the expectation-maximization mechanism allows clusters to have different shapes.

The algorithm has a loose relationship to the k-nearest neighbor classifier, a popular machine learning technique for classification that is often confused with k-means due to the k in the name. One can apply the 1-nearest neighbor classifier on the cluster centers obtained by k-means to classify new data into the existing clusters. This is known as nearest centroid classifier or Rocchio algorithm.

This algorithm is so called k-means as it has k clusters and it is dependent on the two factors firstly the distance factor calculated from the neighbouring clusters and the clustering algorithm which decides which neighbour is the nearest to the data. The data is clustered into the existing cluster whichever is the nearest to it, this is decided by considering the lowest distance factor from all the existing clusters (as seen in figure 2.1). This algorithm is

considered to be one of the best data mining algorithm useful for finding the unseen patterns in the present data and predicting the properties of the new data encountered in the process.

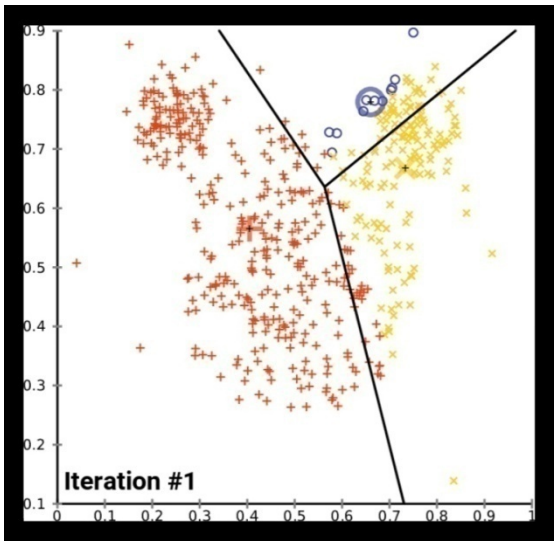

Figure 2.1: Clustering after iteration #1

2.4 Linear Regression

Linear regression is a basic and commonly used type of predictive analysis. The overall idea of regression is to examine two things. It does a set of predictor variables do a good job in predicting an outcome (dependent) variable and second, which variables in particular are significant predictors of the outcome variable, and in what way do they–indicated by the magnitude and sign of the beta estimates–impact the outcome variable? These regression estimates are used to explain the relationship between one dependent variable and one or more independent variables. The simplest form of the regression equation with one dependent and one independent variable is defined by the formula y = c + b*x as shown in Eq 2.1 , where y = estimated dependent variable score, c = constant, b = regression coefficient, and x = score on the independent variable as derived in Eq 2.2 and Eq 2.3. Naming the Variables. There are many names for a regression's dependent variable. It may be called an outcome variable, criterion variable, endogenous variable, or regressand. The independent variables can be called exogenous variables, predictor variables, or regressors.The various forms of linear regression are shown in Eq 2.2 and Eq 2.3

The Dependent and Independent variables used are

$$Y_i = \beta_0 + B_1 X_i + \epsilon_i \quad \text{Eq 2.1}$$

$$\hat{\beta}_1 = \frac{\sum (X_i - \overline{X})(Y_i - \overline{Y})}{\sum (X_i - \overline{X})^2} \quad \text{Eq 2.2}$$

$$\hat{\beta}_0 = \overline{Y} - \hat{\beta}_1 \overline{X} \quad \text{Eq 2.3}$$

2.5 Pearson Correlation

In statistics, the Pearson correlation coefficient (PCC), also referred to as Pearson's r as shown in Eq 2.5 , the Pearson product-moment correlation coefficient (PPMCC) or the bivariate correlation, is a measure of the linear correlation between two variables X and Y. According to the Cauchy–Schwarz inequality it has a value between +1 and −1, where 1 is total positive linear correlation, 0 is no linear correlation, and −1 is total negative linear correlation.

The main relational coefficient and its dependent variables are

$$\rho = \frac{cov(X, Y)}{\sigma_x \sigma_y} \quad \text{Eq 2.4}$$

$$r = \frac{\sum_{i=1}^{n} \left(x_i - \bar{x}\right)(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n} (x_i - \bar{x})^2 (y_i - \bar{y})^2}} \quad \text{Eq 2.5}$$

2.6 Survey

Recommendation system is an information filtering system that seeks to predict the rating or preference a user would give to an item. Recommendation system [1] are of 3 types- Content based, Collaborative and hybrid recommendation system. These are governed by some rules

which come under fuzzy logic[8]. The traditional approach to this problem would be to either base the decision on a word to word referral of courses by peers or to seek the advice of a guidance counselor. Both of these approaches are quite time consuming and might not be absolutely accurate as they are based on individual perspective. The further analysis of review can be done using sentiment analysis. Sentiment analysis finds the attitude of the user according to the review given by him on the proposed course. In the proposed project the accuracy of recommendation can be adjusted so as to give probabilistic recommendation to a certain degree and rest based on the user item matrix. A Pearson correlation is a number between -1 and 1 that indicates the extent to which two variables are linearly related. The Pearson correlation is also known as the “product moment correlation coefficient” (PMCC) or simply “correlation” .Pearson correlations are suitable only for metric variables (which include dichotomous variables). We could also include a hybrid approach and include content based filtering (which depends on the content of the course too) which could further improve the accuracy of the recommendation. For new students who have not taken any courses before, including a knowledge based approach to the existing model would further be beneficial. In the past decade, there has been a vast amount of research in the field of recommender systems, mostly focusing on designing new algorithms for recommendations. An application designer who wishes to add a recommendation system to her application has a large variety of algorithms at her disposal, and must make a decision about the most appropriate algorithm for her goals. Typically, such decisions are based on experiments, comparing the performance of a number of candidate recommenders. There have been many data mining systems developed in education (Romero and Ventura, 2010) and especially on how recommender systems can be utilized for suggesting courses (O’Mahony and Smyth, 2007) or master programs (Surpatean et al., 2012).Most of them use only the actual course content or the curriculum connections (Lee and Cho, 2011) or the performance of students based on their grades (Gogaet al., 2015) or past selections of student courses (Chuet al., 2003) but there is no research work on suggesting courses based on the developing attributes of students.

2.7 Work Done

The Proposed approach can be divided into two working models. The first being the recommender system for which once a user registers by answering certain important questions , the system generates a knowledge base for them . This knowledge base is passed as the features in distance metric which in the proposed framework is *Cosine* metric [4] . The cosine metric of each user with other user is calculated and kept and this is done in *O(n^2)* so the user item matrix data is also taken accordingly . After the distance metric is optimised for the suggested features then comes the classification of users into groups. This is achieved by using clustering algorithm *hierarchal clustering* [4]. It is preferred as the user item data is of small size and hierarchal clustering performs better in such database. The previously calculated Cosine metric is passed to it to separate the users into different classes and hence classification of users into similar groups is achieved the linkage parameter and threshold are adjusted accordingly so as to divide users into the certain number of classes which is 4 in the proposed approach . Now comes the part of suggesting courses. Once a new user is classified into their particular group , the courses for which the members of the group have given a review with a sentiment rating of 4 and above are suggested with the limit being set to 5 recommendations initially . Now coming to the part of sentiment analysis of reviews , the rating for the course reviews which existing users have given are calculated using this sentiment analyser. The analyzer is modelled on Naive Bayes classifier [4] which is shown to have a high efficiency in sentiment analysis models . The ratings are given in a range of 1 to 5 which is done using *multinomial Naive Bayes* .This analyser helps to give rating to reviews of users and intern in helping to select the desired courses . This model is optimised by removing generic words such as course , time etc and adding them in stop words.

The intended project has been initiated with the sentiment analysis of the reviews and labeling them on a scale of 1 to 5 based on the sample review dataset Figure 5.1. The labeling which involved data pre-processing, cleaning, feature selection and making a classifier for the aforementioned purpose. The initial classifier is a naive bayes classifier which has been seen to be having a good accuracy with text processing has been giving an accuracy of 70%

in the specific case of this project. The multinomial naive bayes has been implemented by using the nltk [6] library and first cleaning the reviews using the stop words and the enchant library[5] for identifying reviews only in English. Each review identified was checked for the features which were selected based on the most frequently occurring words in the set of review words. If the intended review contained the feature its score was calculated based on this information. The average score was brought between 1- 5 using modulo technique[3] and hence the labels for any course review not yet labeled can be sent to the classifier and its score based on the features in classifier will get calculated. Once the process of classifying the reviews had reached to an appreciable accuracy the next step of the project included the gathering of the user item matrix data and the main recommendation classifier for a new user. The user item matrix data was needed so as to make a collaborative recommendation system model, as this was intended as a novel model the reviews had to be such that the users who given them were also to give a certain amount of information about themselves so as to form the knowledge base for features for a new user. This process required a certain degree of effort as the matrix was not made available in a single place. For the purpose of the intended project the matrix thus formed is shown in Figure 5.3. The next step once the matrix was available was to deal with the problem of new user. The new user problem has been referred to as the cold start problem and many approaches are available but for the purpose of this project the problem was solved using a set of question which helped in forming a knowledge base for the new user such as their year of study, duration of the course they require, if they require a certificate and so on were put forward to which a users reply determined his/her knowledge base and hence a score could be calculated. The system first uses the metadata of new products while creating recommendations, while visitor's actions go on the second place for a certain period of time. Pearson correlation[4] method in this set was also tested where each user was matched with a reviewer and the tendency of the user when it matched with a reviewer meant this user could also be recommended the same set of recommendation given to the reviewer. The final response involved the use of making of clusters for a set of similar user and then the new user was tested for each cluster.

Chapter 3

System Analysis and Design

3.1 Software Requirement Specification

Software requirement specification (SRS) is the starting point of the software development activity. Little importance was given to this phase in the early days of software development.

The emphasis was first on coding and then shifted to designing. As system grew more complex, it became evident that the goals of the entire system cannot be easily comprehended. Hence the need for the requirement analysis phase arose.

3.1.1 Definition

The large software systems, requirements analysis is perhaps the most difficult activity and also the most error prone. Some of the difficulty is due to the scope of this phase. The software project is initiated by the client's need. In the beginning these needs are in the minds of various people in the client organization. The requirement analyst has to identify the requirements by talking to these people and understanding their needs. In situations where the software is to automate a currently manual process, most of the needs can be understood by observing the current practice. The SRS is a means of translating the ideas in the mind of the clients (the input), into formal document (the output of the requirements phase). Thus, the output of the phase is a set of formally specified requirements, which hopefully are complete and consistent, while the input has none of these properties.

The course recommender system would suggest best suitable courses to the new student after taking some inputs from him/her on some important attributes that will be needed to find the similar user to this new student. This recommendation should be based on the attributes given by the student and should give the best suitable course. The system should also be able to do semantic analysis of the user's reviews and should bifurcate them into good or bad reviews. The system should also be well acquainted with the new course added to various sites, whose users are not available and should identify the course to be good or bad according to the

reviews present on the course. The system should have a UI for the new user, where he or she can enter his or her details and search for the course needed and should also contain an UI which would show the courses recommended.

The system recommends the user the best course from a large no. of courses available on different sites on the same field of interest. It helps the user to save his or her precious time to search over the internet about the course and read thousands of comments and reviews that may be discrete and analogous. Even then the user would not be able to search all the sites and all the courses available on a particular topic. The system maintains a dataset of users and their reviews on different courses on different sites.

The system analyzes the best course taken by a similar group of users on the basis of their reviews on the courses. Then the system takes some fields of inputs from new user to find the similar users existing in our dataset and then suggest the course best reviewed by the similar group of users.

3.1.2 Functional Requirements

The official definition of a functional requirement is that it essentially specifies something the system should do. Typically, functional requirements will specify a behavior or function of the system under the consideration.

- It should ask relevant questions from the user.
- It should be able to take input as the answers provided by the user.
- It should display the list of recommended courses.

3.1.3 Hardware Requirements

Hardware requirements are the requirements needed by the external system using its hardware components so that it is able to run the desired applications within proper response time and with better performance.

- RAM: 2 GB (minimum)
- Storage: 8 GB(minimum)

- System Required: At least Pentium IV processor

3.1.4 Software Requirements

Software requirements are the different software's needed by the system to run the application properly. The application is made using software's and run using these software's smoothly.

- Python IDE(PyCharm, Jupiter Notebook, Spyder)
- NLTK toolkit
- NumPy , Enchant

3.2 Dataset Description

3.2.1 Dataset Extraction

The data set was extracted through a web scrapper based on node which is a JavaScript based runtime environment build on chromes v8 JavaScript engine. As an asynchronous event driven JavaScript runtime, Node is designed to build scalable network applications and using packages like express which is Fast, unopinionated, minimalist web framework for Node.js.

3.2.2 Validation of Dataset

The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration.

The validation set is used to evaluate a given model, but this is for frequent evaluation. This data is used to fine-tune the model hyperparameters. Hence the model occasionally sees this data, but never does it "Learn" from this, it is used for the validation set results and to update higher level hyperparameters. So the validation set in a way affects a model, but indirectly.

The objective was to make sure that test set met the following criteria it was supposed to be large enough to yield statistically meaningful results and is representative of the data set as a

whole getting a dataset with similar characteristics to that of the training set. As our set meets the preceding conditions our goal was to create a model that generalizes well to the new data. The results were not getting high accuracy that might have indicated that the test data leaked into the training set . it was inadvertently trained on some test data as a result we measured accurately about how well our model generalizes to new data.

The Test dataset provides the gold standard used to evaluate the model. It is only used once a model is completely trained (using the train and validation sets). The test set is generally what is used to evaluate competing models (For example on many Kaggle competitions, the validation set is released initially along with the training set and the actual test set is only released when the competition is about to close, and it is the result of the the model on the Test set that decides the winner). Many a times the validation set is used as the test set, but it is not good practice. The test set is generally well curated. It contains carefully sampled data that spans the various classes that the model would face, when used in the real world.

3.2.3 Data Split ratio

This mainly depended on 2 things. First, the total number of samples in your data and second, on the actual model of the training .Some models need substantial data to train upon, so in this case it would optimize for the larger training sets. Models with very few hyperparameters will be easy to validate and tune, so it can probably reduce the size of the validation set, but if the model has many hyperparameters, that would want to have a large validation set as well(although it should also be consider cross validation). Also, if it happens to have a model with no hyperparameters or ones that cannot be easily tuned, probably both dataset are needed.

Here in the considered project model cross validation train/test split uses a 1:4 ratio for test:train data where 20 % data is used for testing and to avoid overfitting the training data is taken to be as random as possible by shuffling the dataset.

3.3 Threats to Validity

Validity is the quality of choices about the particular forms of the independent and dependent variables. These choices will affect the quality of research findings.

3.3.1 Mortality

The extent to which the project can expand to is numerous it might expand it's reach by even suggesting YouTube channels that people should subscribe to or books to read or newsletters or blogs to look forward to and subscribe to. It can even suggest independent teachers who are teaching on web that are not yet included to the database and can even add local education centers.

3.3.2 Maturation

The scope of the project is to extend its reach to people a available website and a app so that people can easily access the program some better program can be found instrumentation not being able to port it into an app and not being able to make it available across all platforms.

3.3.3 System Fails

The system can't provide accurate results for the courses for which user have given no input about, making the system ineffective and bias against systems it is trained on . Users need to input their recommendations that might not be correct as they may not have knowledge of better courses present. Some course owners can try to make their course better by creating multiple users who recommend their course .These system failure scenarios may cause the system to give inaccurate results and hence must be dealt with.

3.4 Use Case diagram

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is

involved. As shown in the figure 3.1, it depicts the way a student interacts with the recommendation system, and how an engineer can optimize the system.

3.4.1 Use Cases

Use Cases are the spherical shaped diagrams which the name of the processes in verb-noun form.

Eg: Login, Book Ticket, Make Payment, View Ticket, etc.

3.4.2 Actors

Actors are the active participants which take part in the process who can either be a person or a proposed system.

Eg: Customer, Server, Cinema Executive.

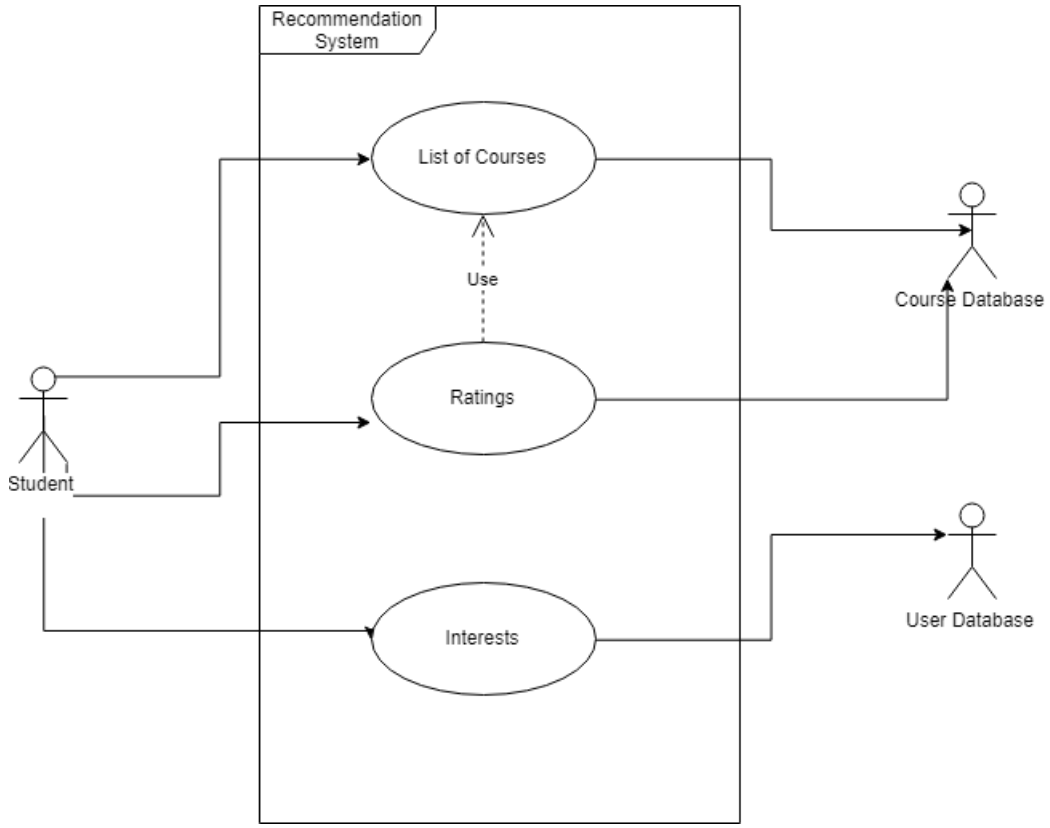

Figure 3.1: Interaction of recommendation system

3.5 DFD

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination. Figure 3.2 shows DFD level 0, which shows low level of detail. Whereas, figure 3.3 shows DFD level 1 which depicts more level of detail as shown below.

3.5.1 DFD Level 0

DFD Level 0 is also called a Context Diagram. It's a basic overview of the whole system or process being analyzed or modeled. It's designed to be an at-a-glance view, showing the system as a single high-level process, with its relationship to external entities. It should be easily understood by a wide audience, including stakeholders, business analysts, data analysts and developers.

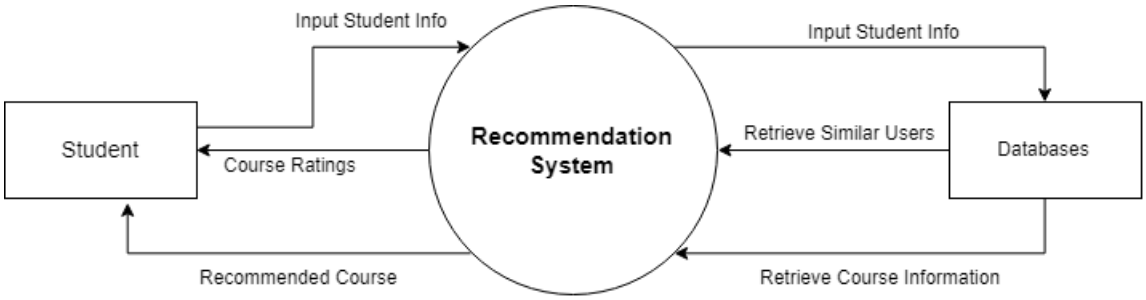

Figure 3.2 DFD level 0

Above figure 3.2 shows the contextual level DFD which shows Recommendation system as whole and emphasizes the way it interacts with external entities. It only contains one process node that generalizes the function of the entire system in relationship to external entities.

3.5.2 DFD Level 1

DFD Level 1 provides a more detailed breakout of pieces of the Context Level Diagram. It will highlight the main functions carried out by the system, as we break down the high-level

process of the Context Diagram into its sub processes that can then be analyzed and improved on a more intimate level.

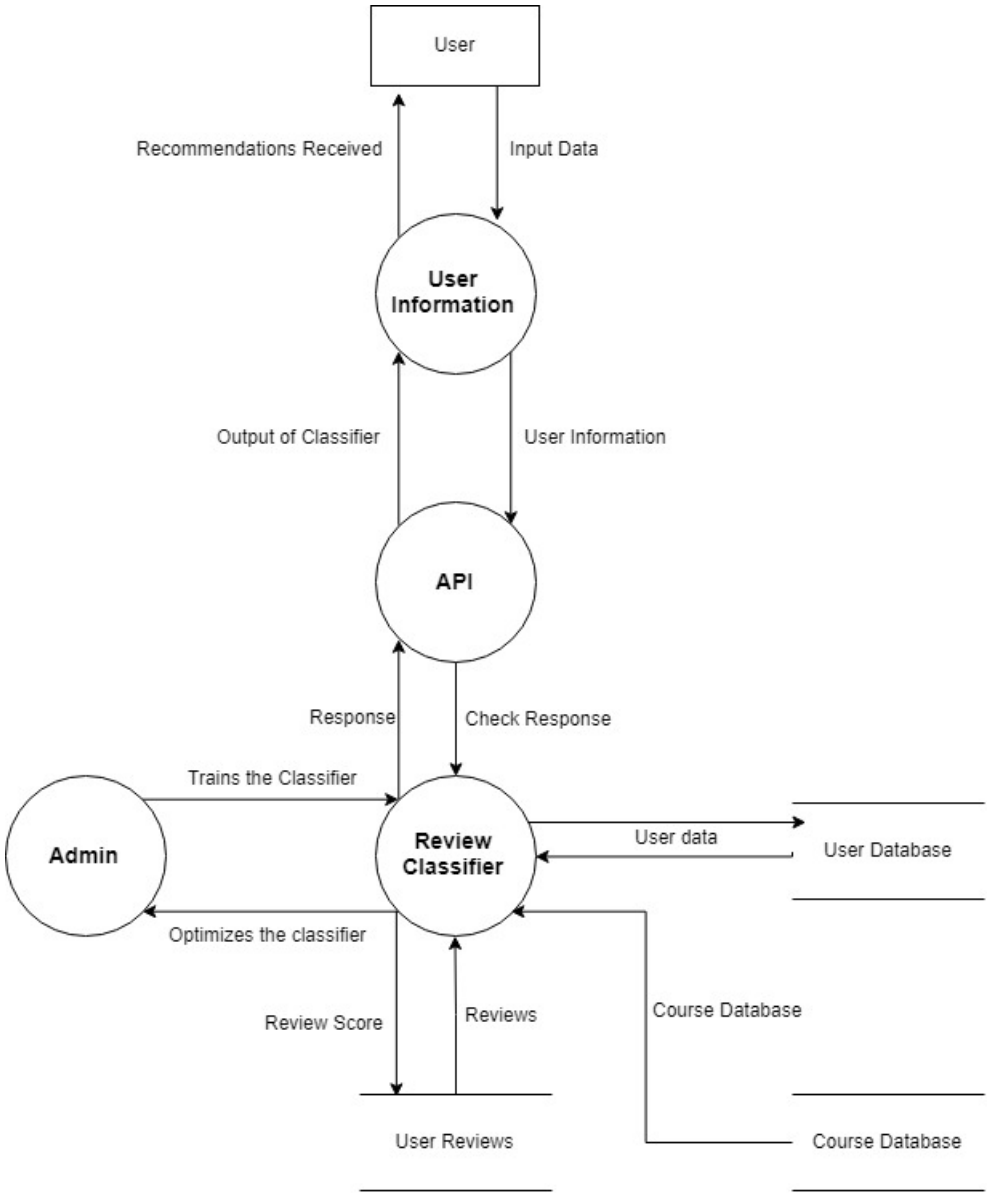

Figure 3.3 DFD level 1

In Figure 3.3, we break up the single process into many sub processes which are part of the Recommender system and how the transforms and moves through these sub processes.

Chapter 4

Proposed work

4.1 Implementation

The course recommender is one of the most powerful tool a student can get his hands on it is one of the tools that is a game changer in terms of being effective and providing results to the student. One of the best things a student can do in his life is to use his time effectively and given proper guidance any student can shoot to the moon. Course recommendation is a special project taken into consideration the problems a student goes through his daily life by fixing it and can be used by any student with a keen interest to learn and grow. As a student we know how big the web really is and in this ocean of information finding the one we need is too tough going throw numerous pages to get what we need is not that effective approach so we tackled our this daily approach and created this product course recommendation system. To learn new things there are multiple websites like Udemy, Coursera etc. Where you can search the course you want that's fine but how to know this one is the best you don't know every website that is there you have not gone throw all the courses present at that website. Ratings can be deceiving and so many courses have so many ratings which among them is for you so we created this project who provides you with the best course for you in whole of the internet which will give you the best result there is.

Being a smart worker is always better than being a hard worker. The intended algorithm also consider how well versed the user is with the subject and also keeps in consideration their older courses taken to recommend the best course there is. The recommended courses can be paid or free.

But for the purpose of this project all courses considered are free courses as most students try to get the audited version for free to get the course material .The used algorithms can be discussed with fair brevity and can be understood from the documentation of the course files included in the project .Most algorithms are standard algorithms and can be implemented with fair ease by including libraries and adjusting their parameters.

4.2 Flow Charts

The Flow of process for training the classifier to rate the courses based on the users intent and reviews. Firstly the dataset for course review with name of each course and their review is included into the program. Once the set of review has been added the next step is to preprocess the dataset and to remove any anomaly present in the data that could cause the system to give inaccurate ratings to the courses . Such as the language of text, null values etc. The next step is to make feature set from given dataset to get the best features so that each review can be correlated to certain words that are important words and that define the intent of the user. Once the featureset of the data has been made next step is to select the classifier to get he optimum result so here in the intended project naive bayes binary classifier has been used for initial purpose so as to rate the data as 1 fo bad and 5 for good. Nlp library has the naive bayes function both multinomial and binary function included so the project used the inbuilt nail bayes function of nltk as the classifier. Once the classifier has been chosen the next step is to divide the data into testing data and training data which can be done with the help of cross validations train test split . For the project the test set taken is divided based on the 4:1 ratio Once the training of the data has been done the classifier was adjusted to achieve ans optimum accuracy with naive bayes and approx 70% was the result the algorithm of naive bayes was able to achieve in the specific case of this project. The intention of the above process is to include as many reviews so that if a user gives a new review it can be labeled and the classifier could be adjusted further to give a adjusted score to each course for getting the best courses and the worst courses based on the selected features of the course review given by the current user.

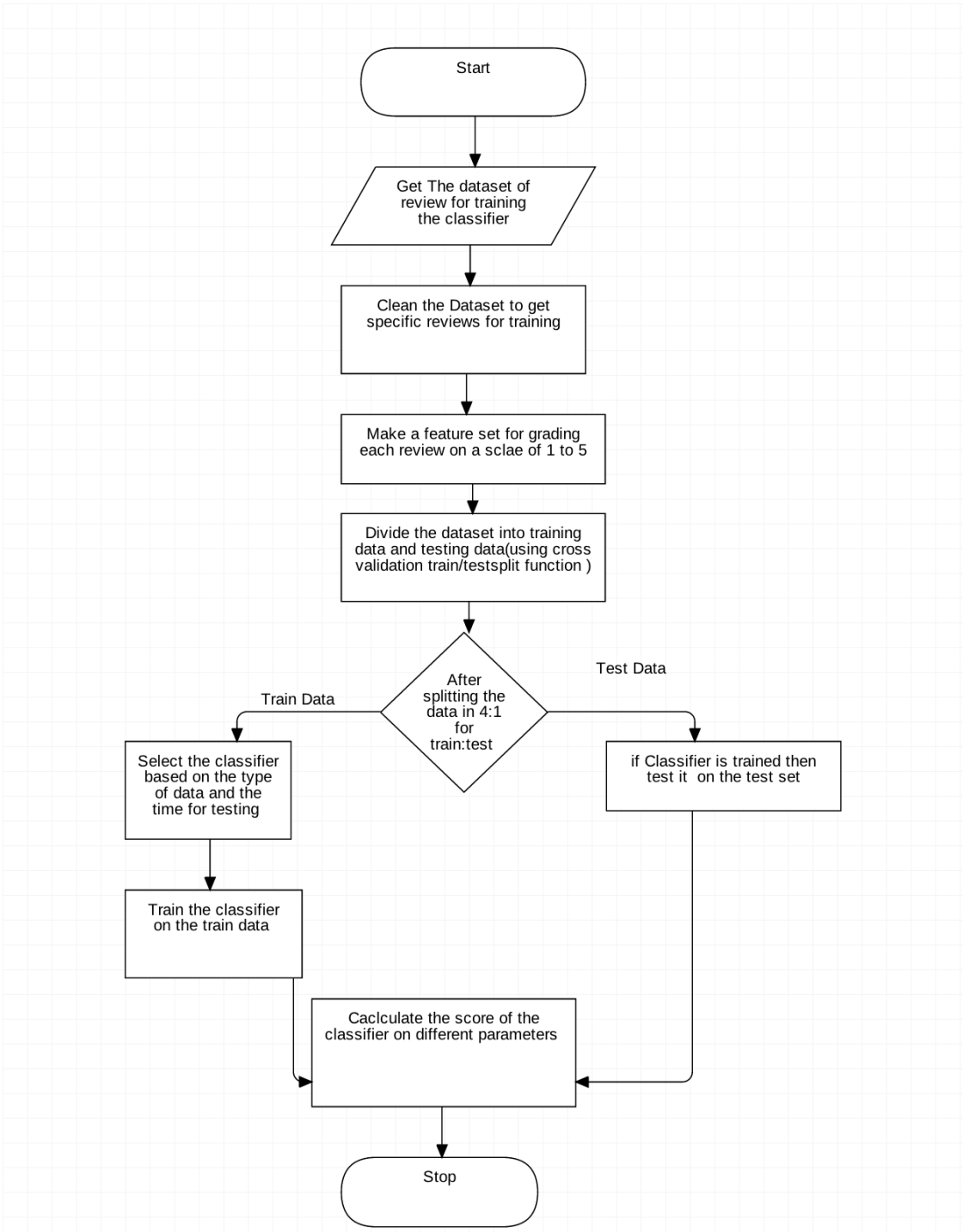

Figure 4.1: Training dataset flow chart

4.2.1 Course of Action

Firstly as the label for each course is available any user needs to login to the system to access his/her data. Once a user registers by answering certain important questions which help the system form a knowledge base for initial recommendation to the new user he/ she can access the recommended courses. Now coming to the backend part it functions based on the collaborative model of recommendation system .Where simlilar users recieve similar recommendations. To achieve the proposed model a user item matrix is made using the data from the previously acquire user data where each user has given review for courses they have done or taken up earlier

Certain key factors which influence the recommendation such as the length of curse needed and the year of study of the student have been taken as the crucial point for making the recommendation. Now the next step is to give a structure or score to each of such user who has given review which has been done by employing the method of average score by giving each feature a score and taking average of all such features. Once the score is caluculated a group or cluster of user has been made by help of kmeans clustering where k has been taken as the number of courses considered for the purpose of training the score classifier of linear regression. A new user can be made to login and his /her score can be calculated based on the points or fatures question and recommendations can be considered based on the closeness to the mean points. K means is as iterative algorithm which intends to group datests based on their closeness to one other or similarity to one another. The recommendation user similarity so they are in true nature with the collaborative model and have been optimised by suggesting the top 5 courses or recommendations to ease the search.

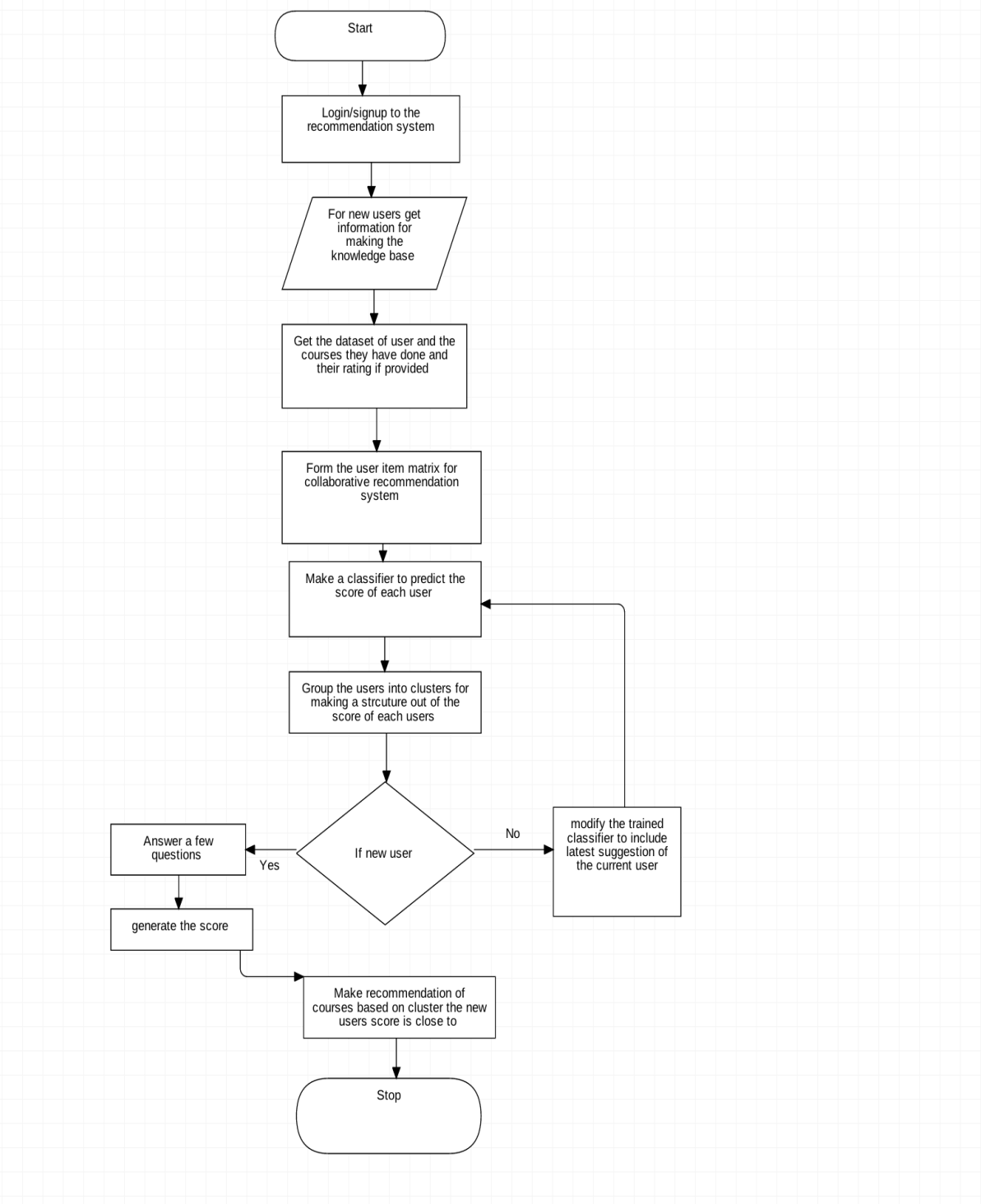

Figure 4.2: Course of action for making a collaborative recommender system

4.2.2 Algorithm

3.1 . Making a sentiment classifier for getting a uniform Score from new Reviews

Step 1 : Get the review DataSet containing the user reviews , the course name , a rating parameter.

Step 2 : To get a random set of course data , shuffle the dataset using *sklearn.utils.shuffle*

Step 3 : Preprocessing the dataSet .

a) Removing Non english reviews.

b) Reindexing the dataset after shuffling .

Step 4 : Making the featureSet .

a) Select most common 3000 words as features .

b) Make a 2d matrix with above selected words as keys and True False as value for each review .

Step 5 : Train the NaiveBayesClassifier using the above created feature sets .

Step 6 : For any new review generate the feature Set and pass it in the classifier for predicting the score

END

3.2. Making the Recommendation System

Step 1 : Get a User Item Matrix with Each user giving reviews about the course they have taken from the list of courses.

Step 2 : Pass Each review into the review classifier and generate a score . Keep the generated score as value of the review.

Step 3 : Covert any Text Field except for reviews into a numerical field by giving each entry a Specific numerical value.

Step 4: For each user Calculate a score using The Score Calculation Formula Definition

a). Year : The year of study of the user among (1,2,3,4)

b). Difficulty Level: The difficulty of the course required (Beginner, Intermediate, Advanced)

c) Length Of Course (in Months) : The time duration of the course needed (1,2,3,4,5)

d) Certification : If certificate is required (Yes, No)

Score = $A/4 + B/3 + C/5 + D$

Step 5: Train The Linear Regression classifier for the above calculated score of each

user . With their input being the four numerical columns and the label being the score .

Step 6: For any new user get the four input fields and calculate the score using the classifier.

Step 7: Get the users with same score as the new User and for the reviews where review Score is high recommend them as recommendation to the user.

END

Chapter 5

Implementation and Results

5.1 Working Explanation

The working project involves the use of jupyter notebook for implementation of each module and the prerequisites mentioned above require nltk.download() to download the various libraries. The loading of dataset into python is helped by the use of pandas and also the numpy library has been used for ease of processing. The datasets after being loaded look like the figure 5.1. Where the reviews have a serial no and the text, and the course id .To deal with the preprocessing nltk library has been used.

```
In [276]: df_sliced
```

Out[276]:

	SNo	CourseId	Review	Label
103701	0	project-planning	Lessons are not compatible with tablets.	4
120046	1	responsive-web-design	excellent effort	5
58682	2	introcss	Colleen is an excellent teacher and all of her...	5
58024	3	intro-programming	Great course! I very loved this class, althoug...	5
113302	4	python-data-analysis	The lectures and explanations need more clarit...	2
36444	5	financial-markets	A lot to learn from this course	5
47422	6	guitar	Though I have learned guitar for many years. T...	5
31451	7	english-principles	This course provides fundamental principles fo...	5
135780	8	web-frameworks	The course is good. I already use Bootstrap an...	3
89840	9	modern-world-2	An amazing trip through modern history!Thanks ...	5
61100	10	java-programming	It is overall an ok course. Samiliarly as a pr...	3
130209	11	technology-of-music-production	Extremely informative. Definitely going to pay...	5
115908	12	python-network-data	it is easy	5
77849	13	machine-learning	It was very information and had good hands-on ...	5
11594	14	bioinformatics-methods-2	introduction to many useful online data resources	4
119619	15	reproducible-research	Excellent course that is both well presented a...	5
94563	16	organizational-behavior	Very Good	5
79996	17	machine-learning	My first coursera course. I've taken others si...	5
5477	18	ancient-marine-reptiles	Awesome	5
116337	19	python-network-data	explanations are very good!!	5
113223	20	python-data-analysis	Great course for data preparation for ML which...	4
67471	21	learning-how-to-learn	I knew that this course was one of the most po...	5
17445	22	clinical-research	A very comprehensive, thorough and complete co...	5
45306	23	global-financial-markets-instruments	Very interesting	5

Figure 5.1 Set of reviews loaded into ipython notebook

5.2 Preprocessing of data

Once the Data has been loaded the next step involves the process of preprocessing the data. This step includes selecting only the English reviews which has been done with the help of enchant library where the first word of a set of 10000 reviews has been selected and checked using the enchant if it was not in English then its index was stored in a array and the set was subtracted from 10000 to get new indexes which contained only English reviews. Preprocessing can be seen in the figure 5.2.

Removing Non English Reviews

In [15]:
```
# from enchant.checker import SpellChecker              # for removing non english text reviews

# def is_in_english(quote):
#     d = SpellChecker("en_US")
#     d.set_text(quote)
#     errors = [err.word for err in d]
#     return False if ((len(errors) > 4) or len(quote.split()) < 3) else True
```

In [232]:
```
import enchant
# dictionary =enchant.Dict("en_US")
def is_in_english_second_approach(quote):            # faster removal of non english reviews
    dictionary =enchant.Dict("en_US")
    return dictionary.check(quote)
```

In [233]:
```
from nltk.tokenize import sent_tokenize, word_tokenize
```

In [234]:
```
# egwords=word_tokenize(shortDf.iloc[0][2])[0]
# egwords
```

In [235]:
```
# word=word_tokenize(shortDf.iloc[0][2])[0]
# word
```

In [236]:
```
print(shortDf.iloc[0][2])
```

```
Lessons are not compatible with tablets.
```

In [237]:
```
# def remove_rows(shortDf):          # to remove review not in english
#     row=shortDf.size
#     for i in range(10000) :
#         word=word_tokenize(shortDf.iloc[i][2])[0] # checking first word of each review to see if its in english
#         if word=="":
#             shortDf.drop(shortDf.index[i])
#         elif is_in_english_second_approach(word)==False:
#             shortDf.drop(shortDf.index[i])
```

Figure 5.2: Function is_in_english_second_approach() which makes only English reviews available

The main feature to use the enchant function rather than using the nltk library inbuilt function is to make a separate list of only English words and to do this faster with creating a new table of only such reviews instead of removing them from current table. Code snippet for the same is shown in the figure 5.3.

```
In [232]: port enchant
          dictionary =enchant.Dict("en_US")
          f is_in_english_second_approach(quote):  # faster removal of non english rev
            dictionary =enchant.Dict("en_US")
            return dictionary.check(quote)
```

Figure 5.3: Enchant function

```
In [238]: wordsforReference=[]
          for i in range(10000):
              wordsforReference.append(word_tokenize(shortDf.iloc[i][2])[0])

          wordsforReference  # getting all first words of review to see if in english

Out[238]: ['Lessons',
           'excellent',
           'Colleen',
           'Great',
```

Figure 5.4: Words of reference

Words for reference is the array which has all the first words which are checked using enchant for getting the index of non English words and to remove them from the total sets. This has been shown in figure 5.5.

```
In [240]: indexes_to_drop=[]                                    # getting the indexes wher
          for i in range(10000):
              if is_in_english_second_approach(wordsforReference[i])==False:
                  indexes_to_drop.append(i)


          indexes_to_drop

Out[240]: [19,
           23,
           27,
           30,
           31,
           32,
           41,
           43,
           45,
           49,
```

Figure 5.5: Indexes to drop non English reviews

```
In [241]: indexes_to_keep = set(range(shortDf.shape[0])) - set(indexes_to_drop)
          df_sliced = shortDf.take(list(indexes_to_keep))                    # i
```

Figure 5.6: New array of English reviews

In this part the non English indexed reviews have been removed from all reviews and a new array of indexes to keep has been made.

5.3 Making the feature set

Once the preprocessing of data is done the next step involved making of the features so to select the features from reviews the most frequent 3000 words form the set of review words were selected as features. To select most common words nltk library inbuilt fuction most_common() and pandas inbuilt function unique() were used. This can be seen in figure 5.3 given below.

Making the feature set of words

```
In [248]: from nltk.tokenize import sent_tokenize, word_tokenize

In [249]: from nltk.corpus import wordnet

In [250]: from nltk.corpus import stopwords
          stops=set(stopwords.words('english'))

In [251]: allCourseReview=[]                              # getting all the course reviews in a single array
          def get_all_course_review(df_sliced):
              for review in df_sliced["Review"]:
                  allCourseReview.append(review)

          get_all_course_review(df_sliced)

In [252]: all_words=[]                                    # tokenizing each review
          def get_words_from_sentence(all_words):
              word=[]
              for sentence in allCourseReview:
                  word=word_tokenize(sentence)
                  all_words.append(word)
          get_words_from_sentence(all_words)

In [253]: all_words[0]
Out[253]: ['Lessons', 'are', 'not', 'compatible', 'with', 'tablets', '.']

In [254]: # documents = []
          # for index in range(df_sliced.shape[0]):
          #     label=df_sliced.iloc[index][3]
          #     sentence=all_words[index]
          #     for words in sentence:
          #         documents.append((words,label))

In [255]: documents
```

Figure 5.7: Process of making the features using the function the get_from_sentence()

```
In [251]: allCourseReview=[]                                    # getting all the course r
          def get_all_course_review(df_sliced):
              for review in df_sliced["Review"]:
                  allCourseReview.append(review)

          get_all_course_review(df_sliced)
```

Figure 5.8: Get all the reviews of considered courses

The above function gets the names of the courses for which reviews have been considered after preprocessing the course reviews (seen in figure 5.8).

```
In [252]: all_words=[]                                          # tokenizing each review
          def get_words_from_sentence(all_words):
              word=[]
              for sentence in allCourseReview:
                  word=word_tokenize(sentence)
                  all_words.append(word)
          get_words_from_sentence(all_words)
```

Figure 5.9: Tokenize each sentence

Tokenizing each sentence and to get words from each sentence the above function get_words_from_sentence has been used which gets a review as an argument and divides it into words and then stores them into words[] python list (as shown in figure 5.9).

```
In [256]: doc=[]                                                # relating each review to
          for index in range(df_sliced.shape[0]):
              label=df_sliced.iloc[index][3]
              currentReview=all_words[index]
              doc.append((currentReview,label))
```

Figure 5.10: Relating words to labels

Once the words have been tokenized they have to be related to a label for the purpose of training to make a classifier which understands to relate each word to a good or bad review context (shown in figure 5.10).

```
In [258]: wordArray=[]                                          # getting individual word
          for sentence in all_words:
              for word in sentence:
                  wordArray.append(word.lower())

          wordArray

Out[258]: ['lessons',
           'are',
           'not',
           'compatible',
           'with',
           'tablets',
           '.',
           'excellent',
           'effort',
           'colleen',
           'is',
           'an',
           'excellent',
```

Figure 5.11: Getting individual words

```
In [259]: clean_words=[]                                                    # removing unne
          stops.update(".", "?", "(", ")", ",", "-", "'", '"',"!","...",";","course",
          for w in wordArray:
              if w.lower() not in stops:
                  clean_words.append(w.lower())
```

Figure 5.12: Removing stop words

To optimize the process of training all the unnecessary words have to be removed which can be done using the nltk.stopwords which include a list of all unnecessary words that can be removed from the text (shown in figure 5.12).

```
In [261]: clean_words = nltk.FreqDist(clean_words)                          # gettin
          clean_words.most_common(15)
```

Figure 5.13: Getting the frequency of clean words

This gives the list of most common words which occur in the selected review and also gives an insight to which words to remove further for consideration (shown in figure 5.13).

```
In [263]: for i in range(3000):                                  # selecting 3000 most com
              word_features.append(most_common[i][0])
```

Figure 5.14: Selecting 3000 words

Selecting 3000 words as the feature set where if any word one of these if present or not present can help identify the sentiment (figure 5.14).

```
In [266]: def find_features(review):                          # it is a kind of heatmap to
              words = set(review)
              features = {}                                    # the dictionary is true f
              for w in word_features:
                  features[w] = (w in words)
              return features
```

Figure 5.15: Function to create a 2D list

This function makes a 2D list or a key value dictionary where if a word is present in the current review it gets true for the corresponding word from the feature set of words else false. The complexity of this algorithm is O(n*m) where n is the size of review and m is the number of feature set words which in the considered case is 3000.

5.4 Training process

Once the feature section is done the training process comes into the picture where naïve bayes has been used from the nltk library so that the reviews get clarified in a category of1 or 5. The cross validation has been used so as to divide the reviews into two sets of 80 % data for training and 20 % for testing . this has been done so as to avoid overfitting of data . The process of training involved 10 min approx . The input for prediction has to be a featureset where a true is there for each word of the features present in the current review and false otherwise. The classifier gives an accuracy score of 71% and the most informative features have been shown in figure 5.4.

```
In [271]: training_set = featuresets[:1500]
          testing_set = featuresets[1500:]

In [272]: classifier = nltk.NaiveBayesClassifier.train(training_set)

In [274]: nltk.classify.accuracy(classifier, testing_set)
Out[274]: 0.71058990760483298

In [74]: classifier.show_most_informative_features(15)       # this shows if a word is 5 it means pretty good course and if a wo

Most Informative Features
                    spent = True              2 : 5      =    136.5 : 1.0
            significantly = True              2 : 5      =    111.7 : 1.0
                     fine = True              2 : 5      =     86.9 : 1.0
                  similar = True              2 : 5      =     86.9 : 1.0
                    waste = True              1 : 5      =     77.9 : 1.0
                 specific = True              1 : 5      =     77.9 : 1.0
                  correct = True              2 : 5      =     62.1 : 1.0
                   answer = True              2 : 5      =     62.1 : 1.0
                mentioned = True              2 : 5      =     62.1 : 1.0
                  suggest = True              2 : 5      =     62.1 : 1.0
                      pay = True              2 : 5      =     62.1 : 1.0
                    times = True              2 : 5      =     62.1 : 1.0
                reviewing = True              2 : 5      =     62.1 : 1.0
                     bugs = True              2 : 5      =     62.1 : 1.0
                   mostly = True              2 : 5      =     62.1 : 1.0

In [75]: classifier.classify(features)        # probably a multiclass classifier would be better for a range in between 1 to 5
Out[75]: 5

In [76]: classifier.classify(features)          # Testing positive and negetives  11th review is negetive and 1 st is positive
Out[76]: 5
```

Figure 5.16: It shows the most informative features that were found during the training of the naïve bayes classifier

```
In [272]: classifier = nltk.NaiveBayesClassifier.train(training_set)
```

Figure 5.17: Nltk inbuilt classifier

Here the classifier used is the nltk's inbuilt classifier naive bayes . This is a probabilistic classifier which works on the above made feature set(Figure 5.17)

```
In [74]: classifier.show_most_informative_features(15)       # this shows if a word is 5 it means pret

Most Informative Features
                    spent = True              2 : 5      =    136.5 : 1.0
click to expand output; double click to hide output
                    fine = True               2 : 5      =     86.9 : 1.0
                  similar = True              2 : 5      =     86.9 : 1.0
                    waste = True              1 : 5      =     77.9 : 1.0
                 specific = True              1 : 5      =     77.9 : 1.0
                  correct = True              2 : 5      =     62.1 : 1.0
                   answer = True              2 : 5      =     62.1 : 1.0
                mentioned = True              2 : 5      =     62.1 : 1.0
                  suggest = True              2 : 5      =     62.1 : 1.0
                      pay = True              2 : 5      =     62.1 : 1.0
                    times = True              2 : 5      =     62.1 : 1.0
                reviewing = True              2 : 5      =     62.1 : 1.0
                     bugs = True              2 : 5      =     62.1 : 1.0
                   mostly = True              2 : 5      =     62.1 : 1.0
```

Figure 5.18: Most informative words

Figure 5.18 shows words that are most informative words which help differentiate a review from a good or bad based on the trained classifier.

```
In [78]: import pickle

In [79]: save_classifier = open("naivebayes.pickle", "wb")              # Saving the current classifie
         pickle.dump(classifier, save_classifier)
         save_classifier.close()

In [80]: classifier_new = open("naivebayes.pickle", "rb")
         classifier = pickle.load(classifier_new)
         classifier_new.close()
```

Figure 5.19: Classifier function

The above code in figure 5.19 can be used to save the trained classifier for future purposes and to save time for further use. This includes saving the pickle file in write mode and then to open it in the read mode to get the saved classifier and use it.

5.5 User item matrix

After the reviews had been classified the next step was to get the user item matrix so as to get a novel model of collaborative recommendation into the picture. The user item matrix is a 2d matrix where the users are the reviewers and item in the intended case are the courses which the users have taken. This matrix helps classify a similar user to them and the process of recommendation gets much simpler. This can be seen in the figure 5.5.

```
In [45]: def getNumericalValueofCertificate(str):
             if(str=="yes"):
                 return 1
             else:
                 return 0
```

Figure 5.20: Store numerical data

To get the training of the classifier done it accepts numerical data so string data is converted into numerical values.

```
In [2]: import pandas as pd

In [162]: datatoRead=pd.read_excel("Untitled spreadsheet (1).xlsx")

In [4]: datatoRead
```

Out[4]:

	Name	Year Currently Studying in	Difficulty level of Course	Length of Course(in months)	Certificate Needed	object-oriented-java'	robotics-motion-planning'	html-css-javascript-for-web-developers'	python-data-processing'	data-scientists-tools'	internet-of-things-communication'	build-data-science-team'	interactive-python-1
0	AKSHITA AGGARWAL	2	beginner	5	yes	1.0	NaN	1.0	1.0	NaN	NaN	NaN	NaN
1	AMAN	3	professional	5	no	NaN	NaN	1.0	NaN	1.0	NaN	1.0	NaN
2	AMIT RAWAT	2	intermediate	2	yes	NaN	1.0	NaN	1.0	1.0	1.0	NaN	NaN
3	ANMOL BISWAS	2	intermediate	1	no	NaN	NaN	NaN	1.0	NaN	NaN	NaN	1.0
4	ASHWIN GIRI	2	beginner	2	yes	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	GAURI VERMA	1	professional	3	no	NaN	1.0	NaN	NaN	1.0	1.0	NaN	NaN
6	HARDIK GOTHWAL	2	beginner	3	no	1.0	NaN	NaN	1.0	NaN	NaN	NaN	1.0
7	MUSKAN PALIWAL	4	beginner	3	yes	NaN	NaN	1.0	1.0	NaN	NaN	NaN	NaN
8	PAWAN THAPA	4	intermediate	1	yes	1.0	NaN	NaN	NaN	1.0	NaN	1.0	1.0
9	PRIYA JAIN	2	professional	4	yes	NaN	1.0	NaN	1.0	1.0	1.0	NaN	1.0
10	ROHIT MEGHWAL	4	intermediate	1	yes	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN
11	SHUBHAM RUSTAGI	4	beginner	2	no	1.0	NaN	1.0	NaN	NaN	NaN	NaN	NaN
12	SIDDHARTH AHUJA	4	beginner	5	yes	NaN	1.0	NaN	NaN	1.0	NaN	1.0	NaN
13	TUSHAR [illegible]	3	intermediate	2	yes	NaN	NaN	NaN	1.0	NaN	1.0	1.0	NaN

Figure 5.21: User item matrix for collaborative filtering

```
In [50]: def getScore(year,DifficultyLevel,LengthOfCOurse,CertificateNeeded):
             sumOfValues=(year/4)+(DifficultyLevel/3)+(LengthOfCOurse/5)+(CertificateNeeded)
             return sumOfValues
```

Figure 5.22: Calculating scores

The score for each user is calculated so as to get a basis for clustering the user who have given the reviews.

5.6 Create a scoring mechanism

Once the user item matrix had been made the next step proposed was to make a set of scoring mechanism for the users but the purpose involved in this case was solved temporarily using the method of averages. The proposed system intended to use hierarchal clustering for users to make a group and the use cosine Distance metric for classifying new users. The knowledge base for the new users was developed by overcoming the cold start problem by asking a set of question from the new users as shown in figure 5.6.

```
In [142]: sliceOfData=datatoRead.iloc[:,2]

In [173]: newDataFiled=pd.DataFrame()

In [174]: newDataFiled['Year']=datatoRead['Year Currently Studying in']

In [175]: newDataFiled['Length']=datatoRead["Length of Course(in months)"]

In [176]: newDataFiled['Difficulty']=datatoRead['NumericalDifficultyLevelOfCourse']

In [177]: newDataFiled['Certificate']=datatoRead['NumericalValueOfCertificateNeeded']

In [178]: newDataFiled.head()
```

Out[178]:

	Year	Length	Difficulty	Certificate
0	2	5	1	1
1	3	5	3	0
2	2	2	2	1
3	2	1	2	0
4	2	2	1	1

```
In [179]: newDataFiled['score']=datatoRead['NewScores']

In [209]: x_train, x_test, y_train, y_test=cross_validation.train_test_split(newDataFiled.iloc[0:,0:4], newDataFiled['score'], te

In [210]: clf.fit(x_train, y_train)
```

Figure 5.23: Set of questions asked from the new user so as to make a knowledge base

5.7 Results

The end result of the recommendation model is to get suggestion based on the aforementioned schema of the things which has been shown in below figure 5.7.

```
In [365]: def mainProgram():
              test=getDetails()
              score=clf.predict(test)
              recommendation=getRecommendation(score[0])
              return recommendation[1:6]

In [367]: mainProgram()

          Enter name achintya
          enter year of study2
          Enter length of course2
          enter level of difficultybeginner
          enter yes if certificate neededyes

          /Users/achintyasarkar/anaconda/lib/python3.6/site-packages/sklearn/utils/validation.py:395: DeprecationWarning: Passi
          ng 1d arrays as data is deprecated in 0.17 and will raise ValueError in 0.19. Reshape your data either using X.reshap
          e(-1, 1) if your data has a single feature or X.reshape(1, -1) if it contains a single sample.
            DeprecationWarning)

Out[367]: ['python-databases',
           'designexperiments',
           'work-smarter-not-harder',
           'finanzas-personales',
           'synapses']

In [ ]:
```

Figure 5.24: Results of recommendation System

```
In [172]: slicedData=datatoRead['Year Currently Studying in']

In [131]: slicedData['LengthOfCourse']=datatoRead['Length of Course(in months)']

          /Users/achintyasarkar/anaconda/lib/python3.6/site-packages/ipykernel_launcher.py:1: Settin
          gWithCopyWarning:
          A value is trying to be set on a copy of a slice from a DataFrame

          See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing
          .html#indexing-view-versus-copy
            """Entry point for launching an IPython kernel.
```

Figure 5.25: Data Slicing

```
In [357]: def getDetails():
              input("Enter name ")
              year=input("enter year of study")
              year=int(year)
              length=input("Enter length of course")
              length=int(length)
              difficulty=input("enter level of difficulty")
              difficulty=getLevelColumn(difficulty)
              certificate=input("enter yes if certificate needed")
              certificate=getNumericalValueofCertificate(certificate)
              return (year,length,difficulty,certificate)
```

Figure 5.26 Function to get details about the user

```
In [348]: def getRecommendation(score):
              scoreTofind=int(score)
              recommendation=[]
              rows=df_sliced.shape[0]
              for i in range(rows):
                  if(df_sliced['Label'].iloc[i]==scoreTofind):
                      recommendation.append(df_sliced['CourseId'].iloc[i])
              return recommendation
```

Figure 5.27 Get recommendation function

```
In [367]: mainProgram()

          Enter name achintya
          enter year of study2
          Enter length of course2
          enter level of difficultybeginner
          enter yes if certificate neededyes

          /Users/achintyasarkar/anaconda/lib/python3.6/site-packages/sklearn/utils/validation.py:395
          : DeprecationWarning: Passing 1d arrays as data is deprecated in 0.17 and will raise Value
          Error in 0.19. Reshape your data either using X.reshape(-1, 1) if your data has a single f
          eature or X.reshape(1, -1) if it contains a single sample.
            DeprecationWarning)

Out[367]: ['python-databases',
           'designexperiments',
           'work-smarter-not-harder',
           'finanzas-personales',
           'synapses']
```

Figure 5.28: Course recommended results

5.8 Web Application

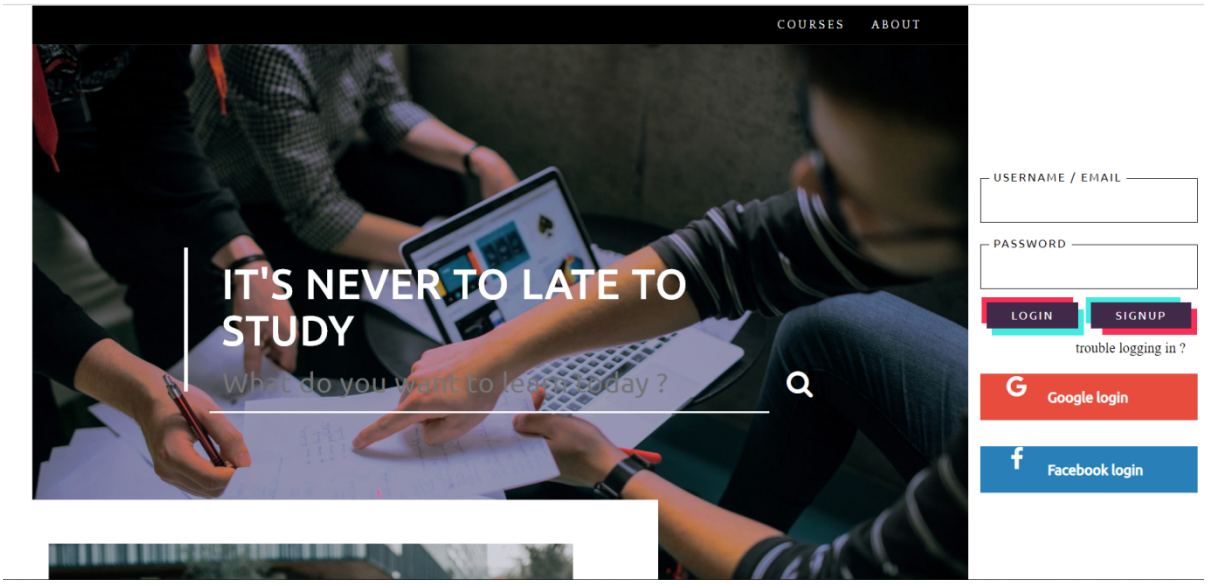

Figure 5.29: Welcome page of web application

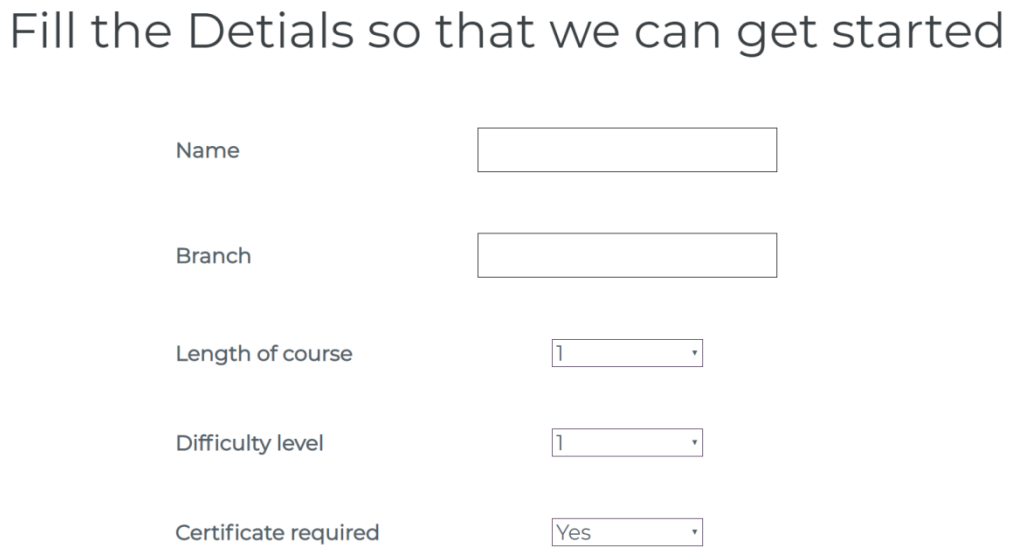

Figure 5.30: Questionnaire

Chapter 6

Conclusion

A recommender system which is able to recommend courses for students would be helpful for students to take wise and well informed decisions while choosing their courses. For our project we have successfully implemented a recommender system for recommending courses given a dataset of ratings provided by various users. This system could provide recommendations for both new users as well as users who have already taken a few courses. We have written a code in python to implement this recommender system which uses collaborative and knowledge based techniques.The main use of the intended project has been achieved to a certain initial degree and this will be helpful in a very strategic manner for students to get the best courses suitable for them and also it will save the time of the users. It is working with the review classifier having accuracy with Naïve Bayes of 74% and the user classifier has accuracy with the basic linear model of 60%. This here shows that many recommendations are being considered by the end users while the probability of a new course being added to their recommendation is also high the system will get better as the database of the users and reviews gains a certain depth in it. This intended project has been developed as a collective effort of the team involved in the research and its first users have also been able to give the recommendation to make the classifier better such that the end product works within certain accuracy aforementioned in the article.This here shows that many recommendation are being considered by the end users while the probability of a new course being added to their recommendation is also high the system will get better as the database of the users and reviews gains a certain depth in it. This intended project has been developed as a collective effort of the team involved in the research and its first users have also been able to give the recommendation to make the classifier better such that the end product works within a certain accuracy aforementioned in the article.

The sentiment analysis model proposed for this approach uses Gaussian Naive Bayes classifier which is initially a binary classifier and in the study of the proposed approach a

multinomial Naive Bayes is seen to have greater impact as binary model just classifies the review as good/bad but a multinomial classifier will be able to classify the review on the scale of 1- 5. The classifier accuracy is 70 percent based on testing data taken from the course dataset. This approach justifies the use of Naive Bayes classifier as it has been seen to have a high accuracy for text-based classification and sentiment analysis [5].

The task of finding similar users, defining features which had a greater impact than others include difficulty level of the course, length of course etc. The initial algorithm using the mentioned features was trained on Linear Regression model. The further study based on the distance parameters led to cosine similarity with a threshold of 0.

The hierarchal clustering model applied uses similarity method as cosine and for the different values of the threshold the number of clusters also varied. The optimum value floats in the range of 0.005 to 0.001, as shown in Table 1, which show the comparison of threshold versus classes formed. The Cosine Similarity distance and Jaccard similarity score were considered appropriate for the proposed approach as these methods show less ambiguity while dealing with the features, especially Cosine similarity at 0.004 threshold, shows an inflection point while applying it at clustering metric.

Distance Metric	**Parameter Name**	**Value**
Cosine_Similarity	Threshold	Classes Formed
	0.001	15
	0.002	7
	0.003	4
	0.004	3
	0.005	3
Jaccard Similarity Score	0.9	36
	0.5	36
	0.01	36
	0.09	36
	1	1

Figure 6.1 :Comparison of Distance Metrics for the Proposed Approach

Chapter 7

Future Work

Once a new user is classified into their particular group , the courses for which the members of the group have given a review with a sentiment rating of 4 and above are suggested with the limit being set to 5 recommendations initially . Now coming to the part of sentiment analysis of reviews , the rating for the course reviews which existing users have given are calculated using this sentiment analyser. The analyzer is modelled on Naive Bayes classifier [4] which is shown to have a high efficiency in sentiment analysis models . The ratings are given in a range of 1 to 5 which is done using *multinomial Naive Bayes* .This analyser helps to give rating to reviews of users and intern in helping to select the desired courses . This model is optimised by removing generic words such as course , time etc and adding them in stop words

The classification which is currently being performed using regression and K-means algorithm. It can be further optimized by using string matching algorithms and giving priority based scores to each feature of the user. Further the accuracy of recommendations can be improved by including content based filtering, which in turns leads to a hybrid recommendation system. The data need to be extracted from various MOOC websites and will compare courses from various courses available on other websites too.

References

[1] A.Tuzhilin and G.Adomavicius, “Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions,” in IEEE Transactions on Knowledge & Data Engineering, vol 17, pp.737-749,2005.

[2] Arxiv Cornell University database of Archived Research : https://arxiv.org [Last Accessed on 28 September 2018]

[3] Github Public Database For Open Projects : https://github.com/achin1tya/NLP [Last Accessed on 25 September 2018]

[4] Sentiment Analysis Tools Research Based Open Articles Platform :

https://medium.com/@datamonsters/sentiment-analysis-tools-overview-part-1-positive-and-negative-words-databases-ae35431a470c [Last Accessed on 25 September 2018]

[5] enchant : https://pypi.org/project/pyenchant/ [Last Accessed on 25 September 2018]

[6] nltk stopwords :https://pythonprogramming.net/wordnet-nltk-tutorial/ [Last Accessed on 2 October 2018]

[7] Naive bayes classifier :https://pythonprogramming.net/naive-bayes-classifier-nltk-tutorial/ [Last Accessed on 2 October 2018]

[8] C.Gupta & A.Jain “Fuzzy logic in Recommender Systems” “Fuzzy Logic Augmentation of Neural and Optimization Algorithms: Theoretical Aspects and Real Applications, pp. 255-273.

Appendix

The data is Having course Review and Labels . Label 1 is for a bad course and 5 is a good course

```
import numpy as np
import pandas as pd

data=pd.read_csv("reviews_by_course.csv")      # getting the data into
python program (Coursera courses with labels)

courses=[]
courses=data["CourseId"].unique()         # saving the course names
courses[1:100]
len(courses)    # getting the number of courses
data.describe()
from sklearn.utils import shuffle
df=pd.DataFrame(data)
df=shuffle(df)    # to randomize the data and select just 10000 rows of  1
lakh rows
data.head()
df.head()

# import random
# random.shuffle(data)
# get only few rows for analysis

shortDf=df[:10000]
shortDf.isnull().sum()                    # check for null values
# shortDf.dropna()                        # to remove null rows
shortDf.insert(0, 'SNo', range(0,len(shortDf)))     # adding a serial no
column
courses=shortDf['CourseId'].unique()               # getting name of all
courses
print(len(courses))
courses
```

Removing Non English Reviews

```
# from enchant.checker import SpellChecker               # for removing non
english text reviews

# def is_in_english(quote):
#     d = SpellChecker("en_US")
#     d.set_text(quote)
#     errors = [err.word for err in d]
#     return False if ((len(errors) > 4) or len(quote.split()) < 3) else
True

import enchant
# dictionary =enchant.Dict("en_US")
def is_in_english_second_approach(quote):             # faster removal of
non english reviews
```

```
    dictionary =enchant.Dict("en_US")
    return dictionary.check(quote)

from nltk.tokenize import sent_tokenize, word_tokenize

# egwords=word_tokenize(shortDf.iloc[0][2])[0]
# egwords

# word=word_tokenize(shortDf.iloc[0][2])[0]
# word

print(shortDf.iloc[0][2])



# def remove_rows(shortDf):          # to remove review not in english
#     row=shortDf.size
#     for i in range(10000) :
#         word=word_tokenize(shortDf.iloc[i][2])[0] # checking first word
of each review to see if its in english
#         if word=="":
#             shortDf.drop(shortDf.index[i])
#         elif is_in_english_second_approach(word)==False:
#             shortDf.drop(shortDf.index[i])


wordsforReference=[]
for i in range(10000):
    wordsforReference.append(word_tokenize(shortDf.iloc[i][2])[0])

wordsforReference       # getting all first words of review to see if in
english

indexes_to_drop=[]                                  # getting the indexes
where review are not in english
for i in range(10000):
    if is_in_english_second_approach(wordsforReference[i])==False:
        indexes_to_drop.append(i)



indexes_to_drop

indexes_to_keep = set(range(shortDf.shape[0])) - set(indexes_to_drop)
df_sliced = shortDf.take(list(indexes_to_keep))                            #
it is much faster to create a new Dataframe than to drop

len(df_sliced)

df_sliced=df_sliced.drop(columns=["SNo"])            # reindexing
df_sliced.shape
df_sliced.insert(0, 'SNo', range(0,df_sliced.shape[0]))
df_sliced.iloc[0]

courses[1:100]
```

Making the feature set

```
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import wordnet
from nltk.corpus import stopwords
stops=set(stopwords.words('english'))
allCourseReview=[]                                  # getting all the course
reviews in a single array
def get_all_course_review(df_sliced):
    for review in df_sliced["Review"]:
        allCourseReview.append(review)

get_all_course_review(df_sliced)

all_words=[]                                       # tokenizing each review
def get_words_from_sentence(all_words):
    word=[]
    for sentence in allCourseReview:
        word=word_tokenize(sentence)
        all_words.append(word)
get_words_from_sentence(all_words)

# documents = []
# for index in range(df_sliced.shape[0]):
#     label=df_sliced.iloc[index][3]
#     sentence=all_words[index]
#     for words in sentence:
#         documents.append((words,label))

doc=[]                                              # relating each review to
its label
for index in range(df_sliced.shape[0]):
    label=df_sliced.iloc[index][3]
    currentReview=all_words[index]
    doc.append((currentReview,label))

wordArray=[]                                        # getting individual
words from reviews
for sentence in all_words:
    for word in sentence:
        wordArray.append(word.lower())

wordArray


clean_words=[]                                                  # removing
unnecesary words
stops.update(".", "?", "(", ")", ",", "-", "'",
'"',"!","...",";","course","'s")
for w in wordArray:
    if w.lower() not in stops:
        clean_words.append(w.lower())

import nltk
clean_words = nltk.FreqDist(clean_words)                                #
getting the most common words
clean_words.most_common(15)

word_features = []
most_common = clean_words.most_common()
```

```
for i in range(3000):                                # selecting 3000 most
common words as features
    word_features.append(most_common[i][0])

len(word_features)

word_features[1:10]

def find_features(review):                         # it is a kind of heatmap to
tell if a review contains any of the most common words
    words = set(review)
    features = {}                                  # the dictionary is true
for words which are present in the review
    for w in word_features:
        features[w] = (w in words)
    return features

features = find_features(all_words[11])#  for a single document


featuresets = [(find_features(review), label) for (review,label) in doc]#
for multiple documents
training_set = featuresets[:1500]
testing_set = featuresets[1500:]
classifier = nltk.NaiveBayesClassifier.train(training_set)

nltk.classify.accuracy(classifier, testing_set)

classifier.show_most_informative_features(15)       # this shows if a word
is 5 it means pretty good course and if a word is 1 it is very bad course

features=find_features(all_words[27])
```

Saving the classifier

```
import pickle
save_classifier = open("naivebayes.pickle", "wb")              # Saving the
current classifier
pickle.dump(classifier, save_classifier)
save_classifier.close()
classifier_new = open("naivebayes.pickle", "rb")
classifier = pickle.load(classifier_new)
classifier_new.close()
```

The classifier gives 1 to reviews with label 1 but it gives 5 to all other reviews as it is a binary class classifier.

```
# #Dictionary of People rating for fruits
```

```
# choices={'John': {'Mango':4.5, 'Banana':3.5, 'Strawberry':4.0,
'Pineapple':4.0},
# 'Nick': {'Mango':4.0, 'Orange':4.5, 'Banana':3.0, 'Pineapple':4.5},
# 'Martha': {'Orange':5.0, 'Banana':2.5, 'Strawberry':4.5, 'Apple':3.5},
# 'Mathew': {'Mango':3.75, 'Strawberry':4.25, 'Apple':3.5, 'Pineapple':
3.0}}
```

Pearson correlation

```
from math import sqrt
#Finding Similarity among people using Eucledian Distance Formula
class testClass():
    def pearson(self, cho, per1, per2):
        #Will set the following dictionary if data is common for two
persons
        sample_data={}
        #Above mentioned varibale is an empty dictionary, that is length =0

        for items in cho[per1]:
            if items in cho[per2]:
                sample_data[items]=1
                #Value is being set 1 for those items which are same for
both persons

        #Calculating length of sample_data dictionary
        length = len(sample_data)
        #If both person does not have any similarity or similar items
return 0
        if length==0: return 0

        #Remember one thing we will calculate all the below values only for
common items
        #   or the items which are being shared by both person1 and
person2, that's why
        #   we will be using sample_data dictionary in below loops

        #Calculating Sum of all common elements for Person1 and Person2
        sum1=sum([cho[per1][val] for val in sample_data])
        sum2=sum([cho[per2][val] for val in sample_data])

        #Calculating Sum of squares of all common elements for both
        sumSq1=sum([pow(cho[per1][val],2) for val in sample_data])
        sumSq2=sum([pow(cho[per2][val],2) for val in sample_data])

        #Calculating Sum of Products of all common elements for both
        sumPr=sum([cho[per1][val]*cho[per2][val] for val in sample_data])

        #Calculating Person Correlation Score
        x = sumPr-(sum1*sum2/length)
        y = sqrt((sumSq1-pow(sum1,2)/length)*(sumSq2-pow(sum2,2)/length))

        if y==0 : return 0

        return(x/y)
        #Value being returned above always lies between -1 and 1
        #Value of 1 means maximum similarity
```

```
def main():
    ob = testClass()

    print(ob.pearson(choices, 'John', 'Nick'))
    print(ob.pearson(choices, 'John', 'Martha'))
    print(ob.pearson(choices, 'John', 'John'))

if __name__ == "__main__":
    main()


datatoRead=pd.read_excel("Untitled spreadsheet (1).xlsx")
```

Trying out the clustering algorithm

```
# import statements
from sklearn.datasets import make_blobs
import numpy as np
import matplotlib.pyplot as plt
# create blobs
data = make_blobs(n_samples=200, n_features=2, centers=4, cluster_std=1.6,
random_state=50)
# create np array for data points
points = data[0]
# create scatter plot
plt.scatter(data[0][:,0], data[0][:,1], c=data[1], cmap='viridis')
plt.xlim(-15,15)
plt.ylim(-15,15)


# import KMeans
from sklearn.cluster import KMeans

# create kmeans object
kmeans = KMeans(n_clusters=4)
# fit kmeans object to data
kmeans.fit(points)
# print location of clusters learned by kmeans object
print(kmeans.cluster_centers_)
# save new clusters for chart
y_km = kmeans.fit_predict(points)


plt.scatter(points[y_km ==0,0], points[y_km == 0,1], s=100, c='red')
plt.scatter(points[y_km ==1,0], points[y_km == 1,1], s=100, c='black')
plt.scatter(points[y_km ==2,0], points[y_km == 2,1], s=100, c='blue')
plt.scatter(points[y_km ==3,0], points[y_km == 3,1], s=100, c='cyan')


# import hierarchical clustering libraries
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering

# create dendrogram
dendrogram = sch.dendrogram(sch.linkage(points, method='ward'))
```

```
# create clusters
hc = AgglomerativeClustering(n_clusters=4, affinity = 'euclidean', linkage
= 'ward')
# save clusters for chart
y_hc = hc.fit_predict(points)


plt.scatter(points[y_hc ==0,0], points[y_hc == 0,1], s=100, c='red')
plt.scatter(points[y_hc==1,0], points[y_hc == 1,1], s=100, c='black')
plt.scatter(points[y_hc ==2,0], points[y_hc == 2,1], s=100, c='blue')
plt.scatter(points[y_hc ==3,0], points[y_hc == 3,1], s=100, c='cyan')


def getLevelColumn(str):
    if(str=="beginner"):
        return 1
    elif(str=="intermediate"):
        return 2
    else :
        return 3

numericalValueOfDifficultyLevel=datatoRead["Difficulty level of
Course”].apply(getLevelColumn)

datatoRead[‘NumericalDifficultyLevelOfCourse']=numericalValueOfDifficultyLe
vel

datatoRead["NumericalValueOfCertificateNeeded"]=datatoRead['Certificate
Needed’].apply(getNumericalValueofCertificate)

from sklearn import linear_model

def getScore(year,DifficultyLevel,LengthOfCOurse,CertificateNeeded):
    sumOfValues=(year/4)+(DifficultyLevel/3)+(LengthOfCOurse/5)+
(CertificateNeeded)
    return sumOfValues


scores=[]
for data in datatoRead:
    scores.append(getScore(data[1],data[16],data[3],data[17]))

for data in datatoRead:
    print(data)

dimension=datatoRead.shape

scores=[]
for i in range (dimension[0]):

sumOfVal=getScore(rows.iloc[i:i+1,1],rows.iloc[i:i+1,16],rows.iloc[i:i+1,3]
,rows.iloc[i:i+1,17])
    scores.append(sumOfVal)

scoretoread=[]
for i in range(dimension[0]):
    scoretoread.append(scores[i][i])
```

```
datatoRead[‘NewScores']=scoretoread

from sklearn.linear_model import LinearRegression

from sklearn import cross_validation
clf=LinearRegression()
x_train, x_test, y_train,
y_test=cross_validation.train_test_split(datatoRead,
datatoRead['NewScores'], test_size=0.2)

clf.fit(x_train, y_train)
y_pred=clf.predict(x_test)
y_pred

slicedData=datatoRead['Year Currently Studying in’]
slicedData['LengthOfCourse']=datatoRead['Length of Course(in months)’]

slicedData[‘NumericalDifficulty']=datatoRead['NumericalDifficultyLevelOfCou
rse']
sliceOfData=datatoRead.iloc[:,2]
newDataFiled=pd.DataFrame()
newDataFiled['Year']=datatoRead['Year Currently Studying in’]
newDataFiled['Length']=datatoRead["Length of Course(in months)”]
newDataFiled[‘Difficulty']=datatoRead['NumericalDifficultyLevelOfCourse']
newDataFiled[‘Certificate']=datatoRead['NumericalValueOfCertificateNeeded']
newDataFiled[‘score']=datatoRead['NewScores']
x_train, x_test, y_train,
y_test=cross_validation.train_test_split(newDataFiled.iloc[0:,0:4],
newDataFiled['score'], test_size=0.2)

def getDetails():
    input("Enter name ")
    year=input("enter year of study")
    year=int(year)
    length=input("Enter length of course")
    length=int(length)
    difficulty=input("enter level of difficulty")
    difficulty=getLevelColumn(difficulty)
    certificate=input("enter yes if certificate needed")
    certificate=getNumericalValueofCertificate(certificate)
    return (year,length,difficulty,certificate)
testcase=getDetails()
showval=clf.predict(testcase)

score=clf.predict(testcase)

def mainProgram():
    test=getDetails()
    score=clf.predict(test)
    recommendation=getRecommendation(score[0])
    return recommendation[1:6]
```