Report Titled

Competitive Problem Classification and Recommendation

Submitted

in partial fulfillment of

the requirements of the degree of

Bachelor of Technology (Computer Engineering)

By

Paras Avkirkar	131070002
Prathamesh Dahale	131070004
Parag Pachpute	131070006
Pranay Patil	131070007
Prathamesh Mhatre	131070009

2016-2017

Under the guidance of

Dr. S.G.Bhirud

Department of Computer Engineering and Information Technology

Veermata Jijabai Technological Institute

Matunga, Mumbai - 400019

2016 - 2017

Veermata Jijabai Technological Institute

Department of Computer Engineering and Information Technology

Certificate

This is to certify that the following students of Bachelor of Technology in Computer Engineering, have completed the report entitled **"Competitive Problem Classification and Recommendation"** to our satisfaction.

Paras Avkirkar	131070002
Prathamesh Dahale	131070004
Parag Pachpute	131070006
Pranay Patil	131070007
Prathamesh Mhatre	131070009

(Name & Signature)
Guide / Supervisor

(Name & Signature)
Head, Department of Computer Engineering and Information Technology

(Name & Signature)
Co-Guide / Co-Supervisor

(Name & Signature)
Director, VJTI

Certificate

The report **"Competitive Problem Classification and Recommendation"** submitted by the following students is found to be satisfactory and is approved for the Degree of Bachelor of Technology (Computer Engineering).

Paras Avkirkar	131070002
Prathamesh Dahale	131070004
Parag Pachpute	131070006
Pranay Patil	131070007
Prathamesh Mhatre	131070009

(Name & Signature)
Supervisor / Guide

(Name & Signature)
Examiner

Date:

Place:

Declaration of the Students

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources.

We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission.

We understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Paras Avkirkar
131070002

Prathamesh Dahale
131070004

Parag Pachpute
131070006

Pranay Patil
131070007

Prathamesh Mhatre
131070009

Date:

Abstract

Identifying class of competitive problem from problem description and other relevant features has been a very important task in competitive programming. In the case of Programming platforms such as Codechef, Spoj etc. classifying problem into a particular algorithm class allows users to solve relatively more difficult problems with ease, explore more related content and in general get more clarity of the problem goal at hand. Tags assigned to problems on such platforms are incomplete or haphazard or ambiguous. Hence, the first part of our project is to build the Class Prediction system which predicts one or more classes of the competitive coding problem. A novice user on such platforms keep on solving easy problems or similar problems he faced in past. This does not ensure the growth or improvement of the user. By recommending a set of problems tailored to user's level and behaviour, which can improve the learning curve of competitive programmer, which will be useful. We propose a Recommendation Engine which recommends problems to the user which encapsulates user's current level, degree of difficulty of the problem and user's ability in each category. Recommendation Engine adapts to user's behaviour as the user gradually improves from novice to expert level.

Hence this thesis classifies the competitive programming problem and recommends relevant problems to user that ensures his or her improvement. Thus this solution will make solving problems on Competitive Coding platforms much easier, will allow the user to streamline his or her efforts and make the learning process enjoyable at the same time.

Contents

List of Figures

List of Tables

1 Introduction

Competitive Programming trace its roots to a competition held by ACM-ICPC at Texas A&M University in 1970. According to (Competitive Programming [1]), Competitive programming is a mind sport usually held over the Internet or a local network, involving participants trying to program according to provided specifications. Contestants are referred to as sport programmers. Competitive programming is recognized and supported by several multinational software and Internet companies, such as Google, Facebook and IBM. There are several organizations who host programming competitions on a regular basis. A programming competition generally involves the host presenting a set of logical or mathematical problems to the contestants (who can vary in number from tens to several thousands), and contestants are required to write computer programs capable of solving each problem. Judging is based mostly upon number of problems solved and time spent for writing successful solutions, but may also include other factors (quality of output produced, execution time, program size, etc.)

There are many competitive coding platforms namely CodeChef [2], Spoj [3], CodeForces [4], HackerEarth [5] which supports over 50 programming languages and has a large community of programmers that helps students and professionals test and improve their coding skills. Its objective is to provide a platform for practice, competition and improvement for both students and professional software developers. Various types of programming contests are held on competitive programming platforms. Commonly there are two type long term and short term contests. Each round of short-term competition lasts from 1 to 3 hours. Long-term competitions can last from a few days to a few months.In most of the competitions, since the number of contestants is quite large, competitions are usually organized in several rounds. They usually require online participation in all rounds except the last, which require onsite participation. The top performers at IOI and ACM-ICPC receive gold, silver

and bronze medals while in the other contests, cash prizes are awarded to the top finishers. Also hitting the top places in the score tables of such competitions may attract interest of recruiters from software and Internet companies.

Programming is a very practical and hands on skill. One has to continuously do it to be good at it. It's not enough to solve the problem theoretically, you have to code it and get the solution accepted. Knowing which algorithm to use and implementing it are two different things. It takes both to be good at programming.

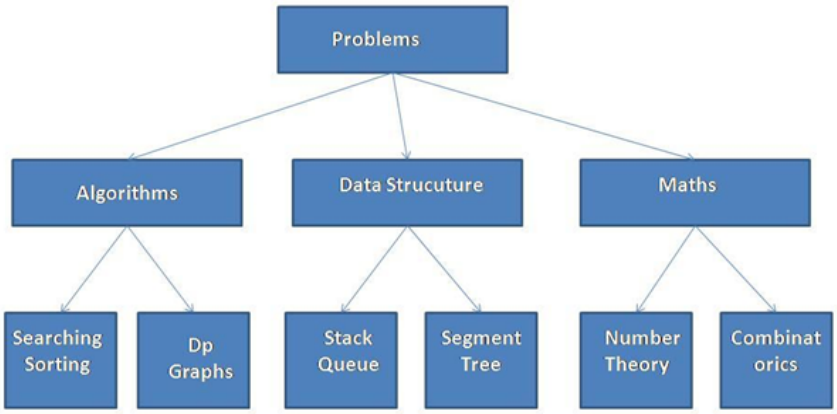


Figure 1.1: Problem Types

Solving basic competitive programming problems requires two things.

1. Intermediate hold on any one programming language
2. Knowledge of basic algorithms in various domains like sort, search number theory, greddy techniques, hashing. More importantly, you have to figure out what, when and where to apply them

It gets really tricky for a novice user. As you can see in Figure 1.1, these are variety of classes of problem. Although to help beginners gain a feeling of confidence many competitive programming platforms provide tutorial series and quality editorials. But reading such editorials is a direct jump from 0 to 1, causing less growth. It would be actually helpful to know the class of problem as a hint so a user can think of the solution.

Programming problems are associated with time limits on execution of the submitted solution. Simply converting English to code doesn't work every time. Programmers need to learn new set of techniques to cope up with time limits. In certain cases Dynamic Programming (DP) comes to the rescue. Many problems can't be solved by just linear data

structures. To solve easy and medium level questions knowledge of graph theory, Disjoint set union, minimum spanning tree is required. In short contests several tough problems can be solved with the knowledge of segment tree, string algorithms, tries, suffix tree. Usually there are not many deep algorithm intensive questions in short contests. However many long contest like monthly challenges can have application of graph coloring, network flow, square root decomposition.

Many programmers argue that the problems in competitive programming do not relate to the real life programming work. For the most part, it is true. But it makes you a better programmer. Time limit always makes you write time efficient solutions. Critical test data helps you write correct solutions, in one go! Further it makes you great at debugging code. Hard problems makes you break down the problem into chunks, solve them individually and bring it all together to solve the main problem. competitive programming is not the only way to master these qualities but it is one of the best ways to do so.

1.1 Purpose

To develop a system to predict class of programming problem and recommend practice problems to user using Machine Learning algorithms. The system would predict class of a problem and would recommend the users which problems to solve which will beneficial to improve him or her.

An expertise in competitive programming is an asset to a programmer, a Computer Science fundamentalist or a mathematician. It takes time and forbearance to enhance ones prowess in this discipline. Most of the times, novice users get perplexed and capitulate before they can acclimatize themselves with different aspects of competitive coding. This often occurs due to the absence of a proper recommendation systems which can streamline the user's understanding.

Competitive Programmer's rationale behind practicing at Competitve Programming platforms:

1. To improve problem solving skills

2. To get hired in product based companies

3. To enjoy and have fun through the solving process

Common problems faced by Competitive Programmers:

1. Identify which problems to solve
2. Identify the class of a problem
3. Identify algorithm needed of a problem
4. Identify difficulty level of problem

The Competitive problem classification and recommendation system perfectly solves the above common problems.

1.2 Problem Definition

Given set of competitive programming problems with their description which includes problem title, description, constraints, submission size, explanation given on not - classify the problems into predefined problem category. Also given set of competitive programmers along with their profile information with submission history which includes no. of problems solved, class of each problem, date of submission, difficulty of problems - recommend set of relevant problems which will improve competitive programmers learning curve.

1.3 Scope

This project consists of predicting tag of given competitive problem and also recommending problem to user for practice. This project will take data from four platforms namely Codechef, Codeforces, Spoj. Class prediction will be implemented for specific set of classes. These classes are chosen based on popularity and data availability of competitive problems. Problems will be recommended to only those users which belong to the platforms mentioned above. Also user will be suggested problems only if he has solved more than 10 problems on that particular platform.

2 Literature Review

This chapter includes survey of current research in the field of text classification and recommendation. Problem classification based on description is first part of this thesis and recommending problems that will improve the problem solving skills of user is the second part. Literature survey in the field of classification is given by section 2.1 and Literature survey in the field of recommendation is given by section 2.2

2.1 Classification

Identifying tags or keywords from text has been a very important class of application of text data [16]. Research has been done on this topic, to use the technique in various application areas, like tagging questions and answers on Stack Overflow, Quora etc. In the case of Questions and Answer sites such as Stack overflow or Quora tagging allows users to explore more related content, build and showcase expertise in a given area and in general get more visibility to the question at hand [16]. In this thesis, we try to apply this technique on competitive programming platforms such as Codechef[2], Spoj[3], Codeforces[4]. In [7], Anudeep talks about various ways in which we can apply machine learning in competitive programming domain. Out of many issues addressed by [7] one which we found very prominent is, "Currently problems are tagged manually and there are multiple issues with this. Tags are high level. They tell if a problem has math involved or binary search involved but does not give us any information about what in math or how complex it is. There are a lot of problems without tags" [7]. In this thesis, one of our objective is to address this particular issue.

There are several approaches to text categorization, such as decision rule based [8], knowledge base based, text similarity based and so on [9]. In this thesis we focus on text categorization using text similarity, in that too more emphasis is given to Keyword extraction. Keyword extraction is a topic that has been generating increasing interest both in industry

and academia [18]. Two approaches were identified for tag prediction for the questions, one was to build a global multiclass classifier and then use method such as One vs All to select the final class. The second approach was to build a discriminant classifier for each of the tags and then predict the final tags choosing the most likely tags [16]. Since we need to predict more than one tags for each of the question, it was decided to use the second approach.

To identify features from the problem description mainly two approaches were identified. First, using Frequent Pattern Mining and second, using Term Frequency Inverse Document Frequency (TF-IDF). Term frequency is the simplest measure to weight each term in a text. In this method, each term is assumed to have importance proportional to the number of times it occurs in a text [10]. While term frequency concerns term occurrence within a text, inverse document frequency (IDF) concerns term occurrence across a collection of texts. The intuitive meaning of IDF is that terms which rarely occur over a collection of texts are valuable. The importance of each term is assumed to be inversely proportional to the number of texts that contain the term [11]. In Frequent Pattern Mining approach we find occurrence frequency of each word, then based on percentage occurrence of words we decide which words to take as features. Detailed procedure regarding Frequent Pattern Mining approach is explained in subsection 6.1.1

2.2 Recommendation

Recommendation algorithm actually is a method to connect users and items, which is at first calculate the user's property model, and then calculate the item's property model, and then the item is recommended to the user by different methods [12]. Collaborative filtering recommendation and content-based recommendation are the two most widely recommendation algorithms for research and application. The idea of content-based recommendation algorithm is based on the item that user has chosen, and then choose the item with similar characteristics from the candidate recommendation item as the recommendation results. However, collaborative filtering gets the items on the explicit or implicit information via the collection of user's past behavior, according to the user's preference for items found the correlation of items by itself, and then based on these correlations to recommend, collaborative filtering focuses on the user's preferences or the ratings of items [13].

Here in this thesis requirement was that, recommendations should be such problems,

that will improve the problem solving skills of competitive programmer. For that reason alternate recommendation strategy using Word2Vec is used. First recommendations are generated from Word2Vec and then problems are filtered, so that only those problems are recommended to user, which will improve his problem solving skills. According to [thomas et al], word2vec is distributed representations of words in a vector space that help learning algorithms to achieve better performance in natural language processing tasks by grouping similar words. Recently, Mikolov [22] introduced the Skip-gram model, an efficient method for learning high-quality vector representations of words from large amounts of unstructured text data. Unlike most of the previously used neural network architectures for learning word vectors, training of the Skip- gram model does not involve dense matrix multiplications. This makes the training extremely efficient.

In this thesis word2vec with skipgram model is used for recommendation. Here the successful submissions of user are considered as the input sentences, and model is trained on these sentences. Word vectors are generated for each of the problem in the sentence. Later when new user arrives and asks for problems to solve, at that time problem recommendation is done using this trained model and previous submissions of that user. Detailed procedure regarding Recommendation using Word2Vec is explained in section 7.2

3 Present Investigation

The core solution for Competitive Program Classification and Recommendation consists of two pillars: Feature and Models. In this chapter we display our research and study about features and models.

3.1 Feature Investigation

For building out Problem Class Prediction Model and Problem Recommendation Model, features are important in machine learning. Which Features of the Problem/User? Number of such features? - govern the model built by machine learning algorithm. Hence, it is also important to analyze and decide the features to be pass for training the data. Consider, a set of problems and user profiles. By observing parameters available on those pages, we can establish a rough observations to decide the parameters that change with the problems and the users. In the Table **??** that summarizes those observations.

3.2 Model Investigation

In this section different algorithms that can be used for text classification and problem recommendation are considered. For text classification K-nearest neighbour and Support Vector Machine are considered, these are explained in subsection 3.2.1 and subsection 3.2.2 respectively. For problem recommendation Collaborative Filtering algorithm is considered, and is explained in subsection 3.2.3

3.2.1 K-Nearest Neighbor

KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. The input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression: In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor. As, you can see in Figure 3.1, the point with star would marked as member of Class B, for k = 2 and member of Class A, for k = 6. In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

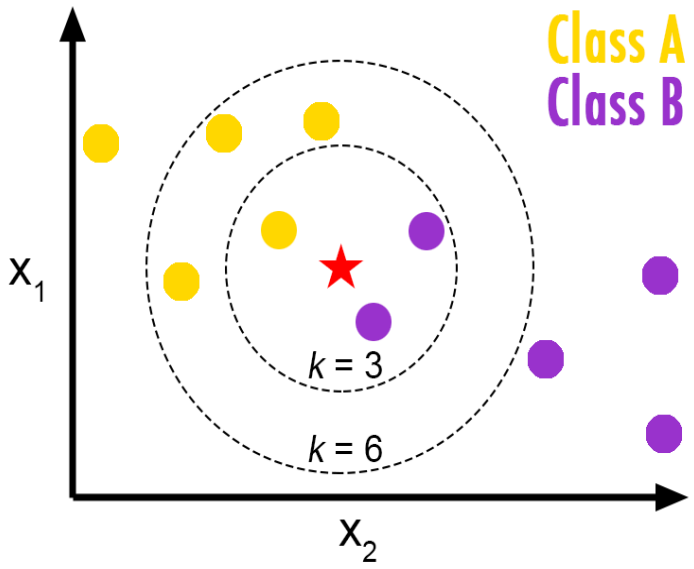


Figure 3.1: K-Nearest Neighbor [23]

In case of competitive coding problems, the features like - words in descriptions, submission size, example given or not, become the axes in the multi-dimensional plot. Then depending on the length of different data points representing problems, knn as a classifier calculates length vector to k-closest neighbours. Then based on the classes of those neighbours, a class is selected based on majority votes. KNN is a robust model for text based classification.

3.2.2 SVM

"Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyperplane that differentiate the two classes very well, as you can see in Figure 3.2. For algorithm class prediction, SVM would

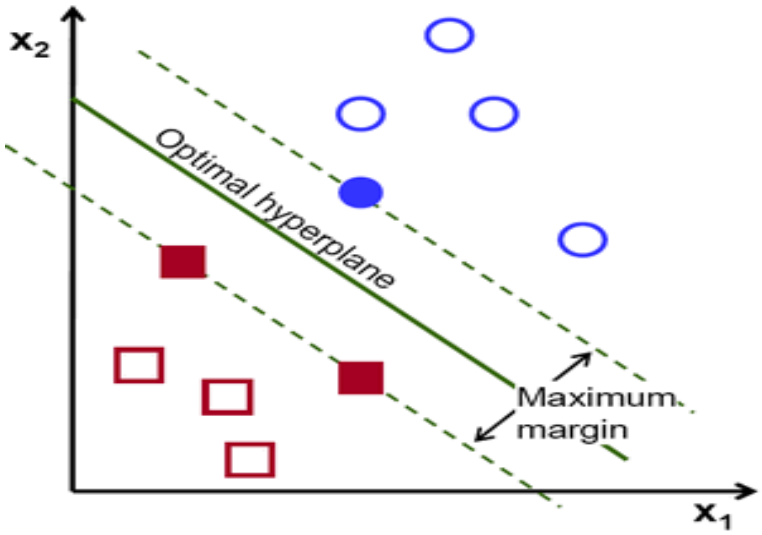


Figure 3.2: SVM [24]

prove to one of the good tool for classifying a competitive coding problem into an decision - algorithm class A - yes, or an algorithm class A - no. It is easier to get an intuition when we have features like - tokens from the problem description, constraints on the problem input, presence of sample example, submission size of the solution.

So, the features draw themselves on a 4-dimensional space and a hyperplane classify the support vectors into two decision classes for each algorithm class. Whether an algorithm belongs to a particular class or not. We can get an estimate of margin of the hyperplane for each algorithm class decision, when we run a new problem vector to predict. Then, for final prediction we can simply take that algorithm class decision for which its hyperplane have the largest margin from the new problem vector.

3.2.3 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 typical flow in collaborative filtering is shown in Figure 3.3 . 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. Collaborative filtering will prove helpful for our second component to help user to climb over the learning curve.

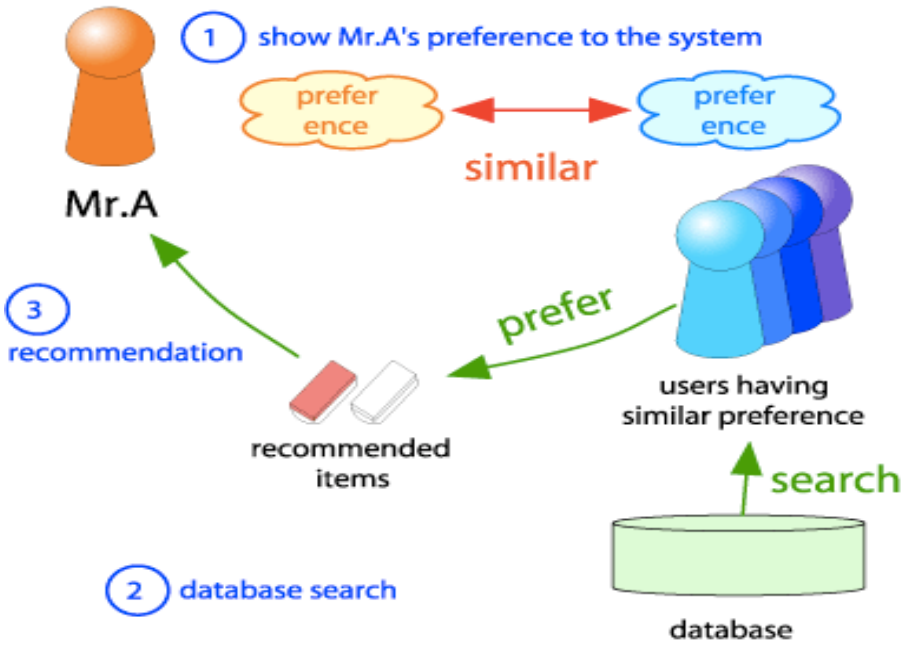


Figure 3.3: Collaborative Filtering [27]

Cosine similarity or Pearson Correlation will helpful to determine similarity between similar competitive coders. We prefer Cosine similarity because it will still work with Sparsity Problem that we face from extracted data (This problem occurs when the amount of items become very large reducing the number of items users have rated to a tiny percentage. In such a situation it is likely that two people have few rated items in common making the correlation coefficient less reliable.). We believe performance from Cosine similarity on vectors would be more accurate than Pearson Correlation. We consider using at this stage using Item-Item based approach rather than User-User based because, items over a dataset can be

easily generalised rather than users. As users have distinct tastes.

3.2.4 Neural network

Neural networks are typically organized in layers. Layers are made up of a number of interconnected 'nodes' which contain an 'activation function'. Patterns are presented to the network via the 'input layer', which communicates to one or more 'hidden layers' where the actual processing is done via a system of weighted 'connections'. The hidden layers then link to an 'output layer' where the answer is output. Neural networks are notable for being adaptive, which means they modify themselves as they learn from initial training and subsequent runs provide more information about the world. A sample neural network topology can be seen inside Figure 3.4. The most basic learning model is centered on weighting the input streams, which is how each node weights the importance of input from each of its predecessors. Inputs that contribute to getting right answers are weighted higher. Neural

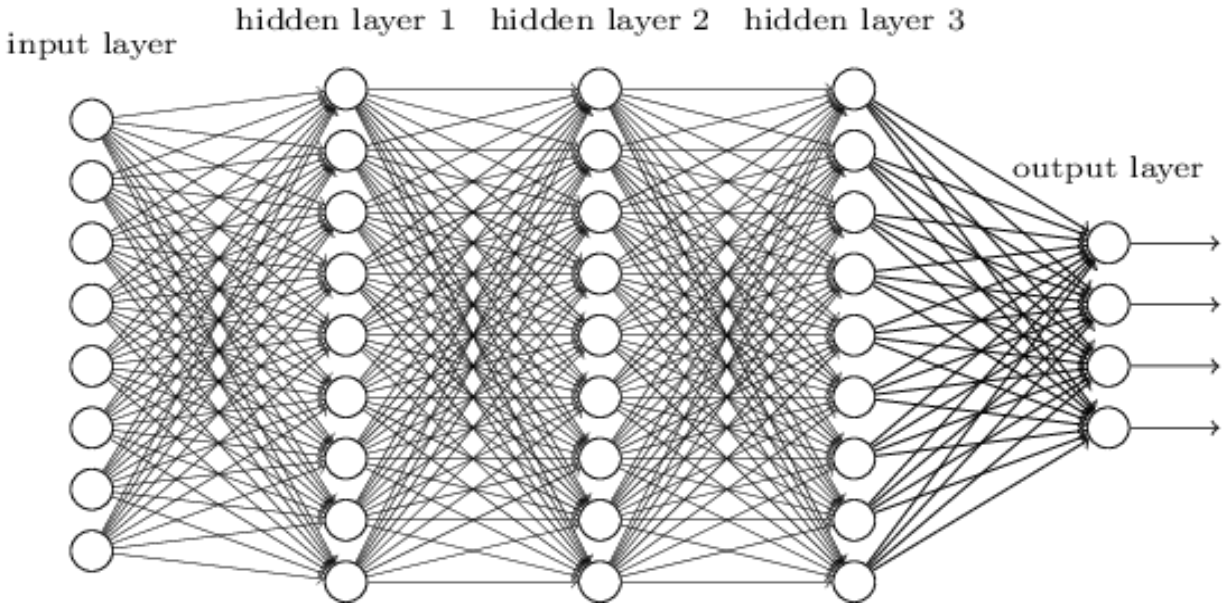


Figure 3.4: Neural Network [25]

networks are preferred when the relationships between variables are vaguely understood or, the relationships are difficult to describe adequately with conventional approaches.

In Figure 3.5 you can see how input to a neuron is calculated based on weighted inputs from neurons of previous layer and bias present at the current neuron. Activation

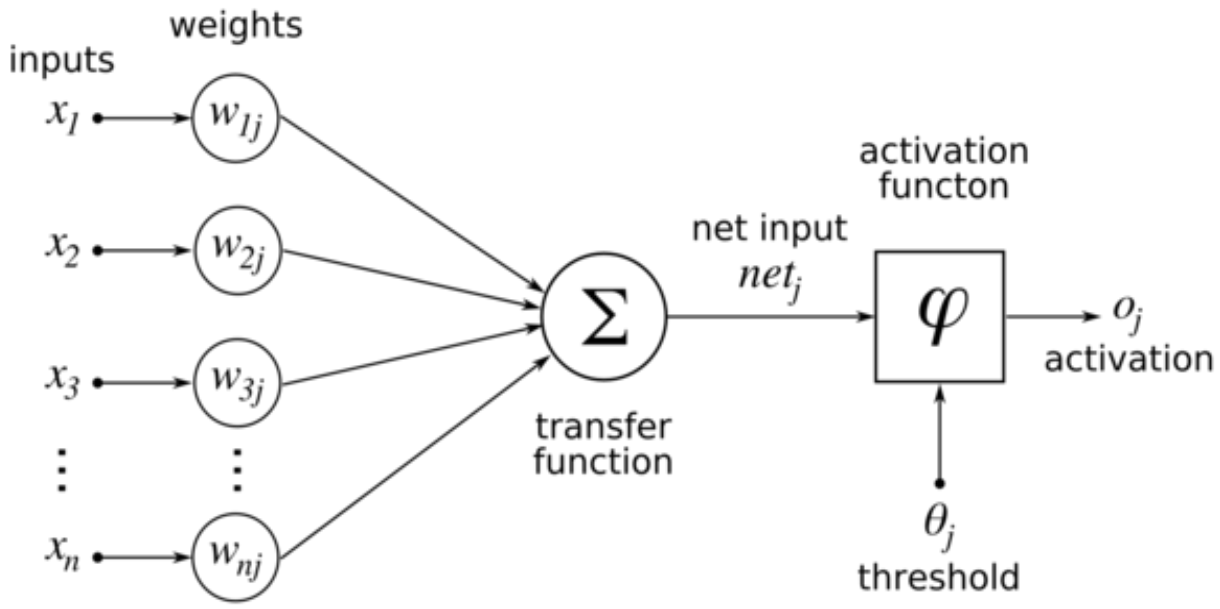


Figure 3.5: Input Output Transition at Neuron [26]

function is being served with the result of this input for calculating output to be sent at neuron in next layer.

We have features as tokens from the problem description, constraints on the problem input, presence of sample example, submission size of the solution. These are applied to the input layers. The output layer has k no of nodes out of which only one will will have output value of 1 and rest 0. No of hidden layers initially we start with two, and later more layers can be added to increase accuracy and precision. The data is divided into two groups, training and testing. Training data is passed to the first layer and the output of the final layer is compared with the actual output. The model is trained with adjusting the weights using backpropagation. We start with lower values of epoch, which can be incremented if need for such arises.

3.3 Precision and Recall

In pattern recognition and information retrieval with binary classification, precision (also called positive predictive value) is the fraction of retrieved instances that are relevant, while recall (also known as sensitivity) is the fraction of relevant instances that are retrieved.In Figure 3.6 you can see diagramatic expression of precision and recall.

Both precision and recall are therefore based on an understanding and measure of rel-

evance. Interpretation of Precision and Recall in this system is as follows:

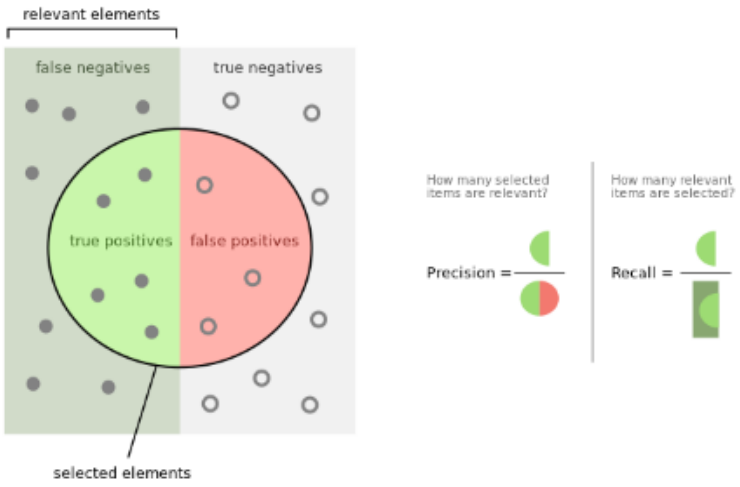


Figure 3.6: Precision And Recall [28]

If system predicts 10 out of 50 problems as DP out of which only 7 are DP and in those 50, 16 problems are of DP then

Precision : 7 / 10 = 0.7

Recall : 7 / 16 = 0.43

4 Planning and Design

Every project works well when it is planned and executed well. In this chapter we chart out high level design (architecture) for the Competitive Problem Classification and Recommendation System. We plan the tasks that need to be carried out to developing the architecture and design the timeline for it.

4.1 Project Architecture

The architecture of the project is a set of components that needed to establish the Problem Class Prediction Model and Problem Recommendation Model from the data at Disparate Coding Platforms as shown in 4.1.

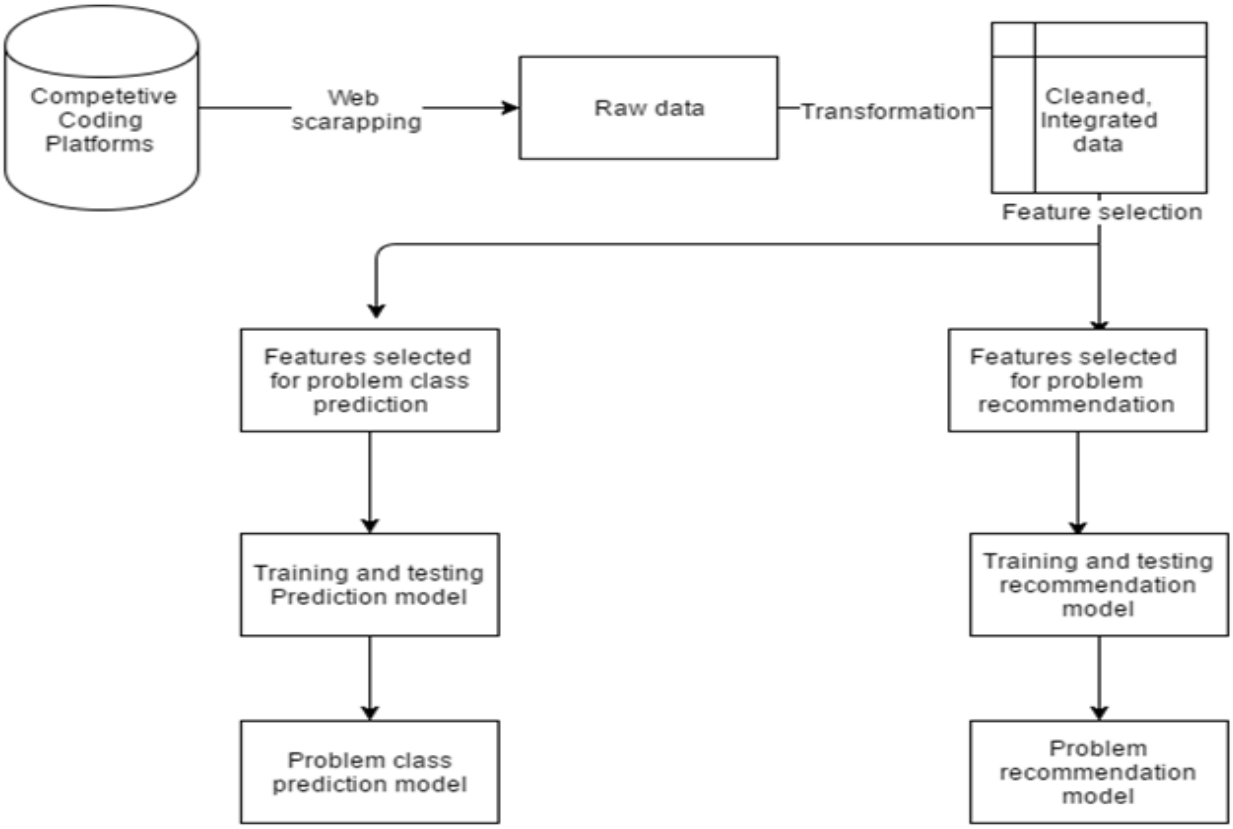


Figure 4.1: Competitve Program Classification and Recommendation Architecture

The architecture of the recommendation system is a set of components that needed to establish the recommendation engine. Figure 4.2 summarizes all those milestones

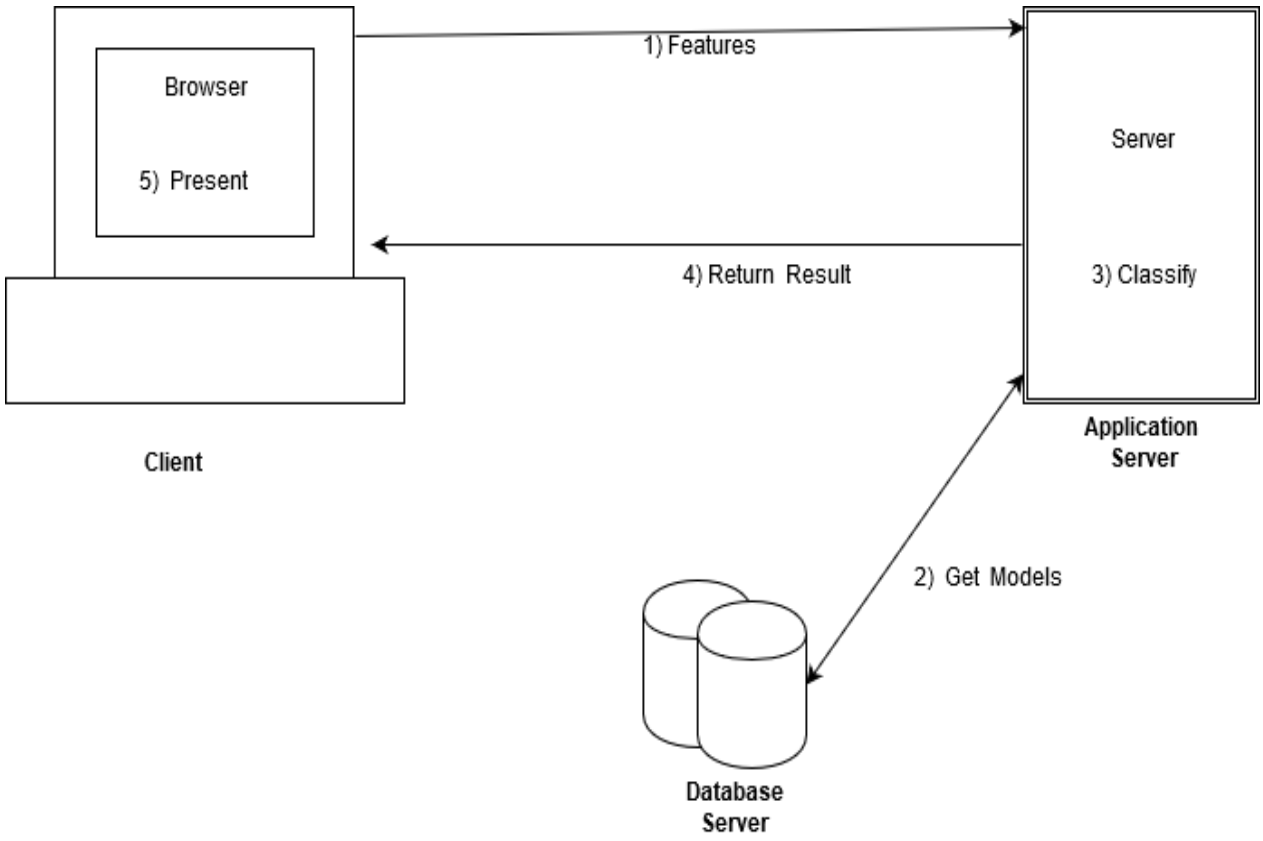


Figure 4.2: Recommendation System Architecture

4.2 Task Set

The project flow mentioned in previous section contains abstract milestones. Drilling down, there are activities that need to be carried out to traverse to a milestone and from there on. All those activities can be grouped into a variety of task. Thus, building a task set. Task Set in Figure 4.3 defines those collection of task for completion of the Problem Classifier and Recommendation Tool

# ▲	↓↑	Summary	Group	Status	Type	Priority	Actual Hours	Estimated Hours
8	☆	Define data extraction rules	Development	To Do	Improvement	Normal	8.00	10.00
9	☆	Planning and ETL initiative	Development	To Do	Improvement	Normal	9.00	7.00
10	☆	Completion and validation of ETL activities	Development	To Do	Improvement	Normal	0.00	10.00
11	☆	Identifying features	Development	To Do	Improvement	Low	0.00	13.00
12	☆	Identifying various applicable models and algorithms	Development	To Do	Improvement	High	0.00	10.00
13	☆	Training those models and comparing accuracies	Development	To Do	Improvement	High	0.00	7.00
14	☆	Design of interfaces for the end user application	Development	To Do	Improvement	Normal	0.00	10.00
15	☆	Implementation of interfaces and unit testing	Testing	To Do	Improvement	Normal	0.00	7.00
16	☆	Integration Testing	Testing	To Do	Improvement	Normal	0.00	10.00
17	☆	Deployment and user feedback	Development	To Do	Improvement	Normal	0.00	7.00

Figure 4.3: Task Set

4.3 Timeline

In previous section, we summarize the activities that needed to be carried out. In current section, those activities are ordered by sequence to be done. Along with the order the time at which activities must be carried out matters. A sufficient time is allotted for the activities. With this following, Figure 4.4, time-line is chart out:

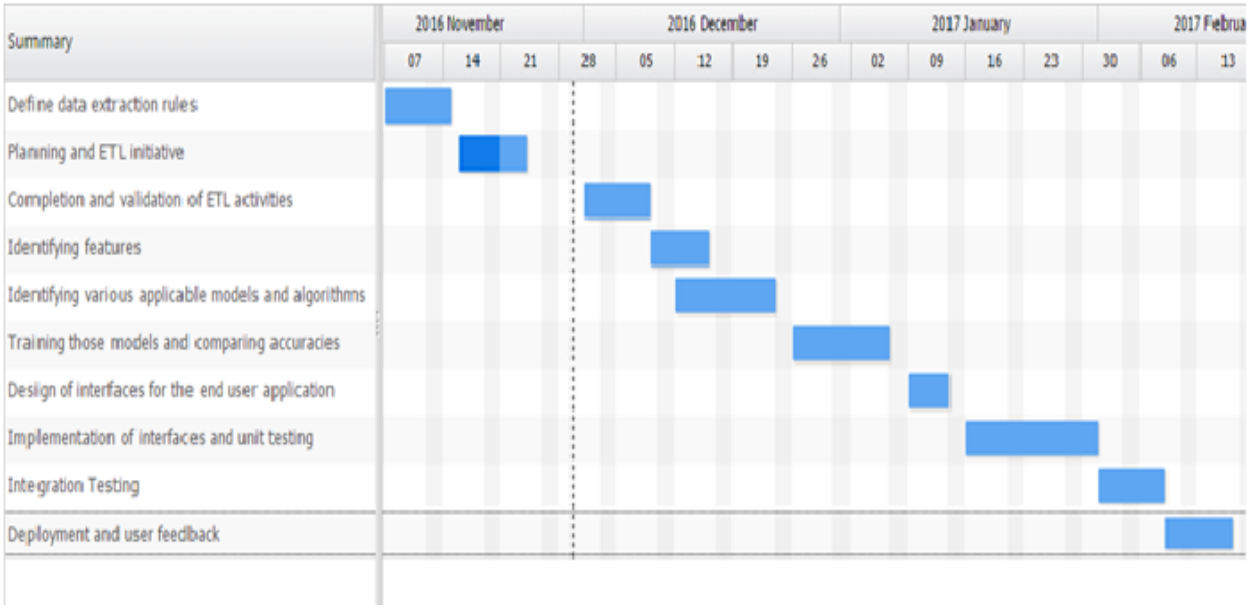


Figure 4.4: Time-line

5 Data Processing

Data Processing contains data collection-cleaning stages. For integration we simply use SQL to integrate cleansed data from different sources. Each of the stages are explained in coming sections.

5.1 Data Collection

For our models, we scraped the data through different platforms. We show different data preprocessing technique to conform the data into suitable format needed by model learning phase.

5.1.1 Sources

We have chosen 4 platforms to extract our data form

1. Codechef

   CodeChef [2] was created as a platform to help programmers make it big in the world of algorithms, computer programming and programming contests. At CodeChef we work hard to revive the geek in you by hosting a programming contest at the start of the month and another smaller programming challenge in the middle of the month. We also aim to have training sessions and discussions related to algorithms, binary search, technicalities like array size and the likes. Apart from providing a platform for programming competitions, CodeChef also has various algorithm tutorials and forum discussions to help those who are new to the world of computer programming.

2. Codeforces

   Codeforces [4] is a Russian website dedicated to competitive programming. It was

created and is maintained by a group of competitive programmers from Saratov State University led by Mikhail Mirzayanov.

3. Hackerearth

   HackerEarth [5] is a startup technology company based in Bangalore, India that provides recruitment solutions.Its clients include Adobe, Altimetrik, Citrix Systems, InMobi, Symantec and Wipro. It is the fourth company to be incubated by AngelPrime, and received US$500,000 for its seed funding round in 2014.Founded in November 2012 by Indian Institute of Technology Roorkee alumnus Sachin Gupta and Vivek Prakash,HackerEarth was a finalist in the 2014 Seedstars World startup competition held in Geneva, Switzerland.It has a competitive programming platform which supports over thirty two programming languages (including C, C++, Python, Java, and Ruby).

4. Spoj

   SPOJ (Sphere Online Judge [3]) is an online judge system with over 200,000 registered users and over 20,000 problems. Tasks are prepared by its community of problem setters or are taken from previous programming contests. SPOJ allows advanced users to organize contests under their own rules and also includes a forum where programmers can discuss how to solve a particular problem. The solution to problems can be submitted in over 40 programming languages, including esoteric ones, via the Sphere Engine. It is run by the Polish company Sphere Research Labs. The website is considered both an automated evaluator of user-submitted programs as well as an online learning platform to help people understand and solve computational tasks. It also allows students to compare paradigms and approaches with such a wide variety of languages.

5.1.2 Extraction Process

Competitive coding platfforms mentioned in previous section does not provide us with API's for direct data extraction. We have to scrape through thousands of web pages to collect required data.

We have devised following algorithm to perform scraping and parsing, this algorithm is depticted in figure 5.1. And the flow of the data for this extraction process is depicted in Figure 5.2

1. Open selenium driver to get access to the web browser
2. Programmatically load the page of a particular platform
3. Navigate to the required entities page using API calls
4. If a pagelet has not been loaded perform a wait system call for few seconds, so as to avoid runtime changes in DOM objects
5. Parse the web page to isolate the data contents
6. Transform the data contents as appropriate to the data types attribute
7. Commit the data loads to the databases
8. Repeat from step 3 until all data is acquired

Figure 5.1: Extraction Algorithm

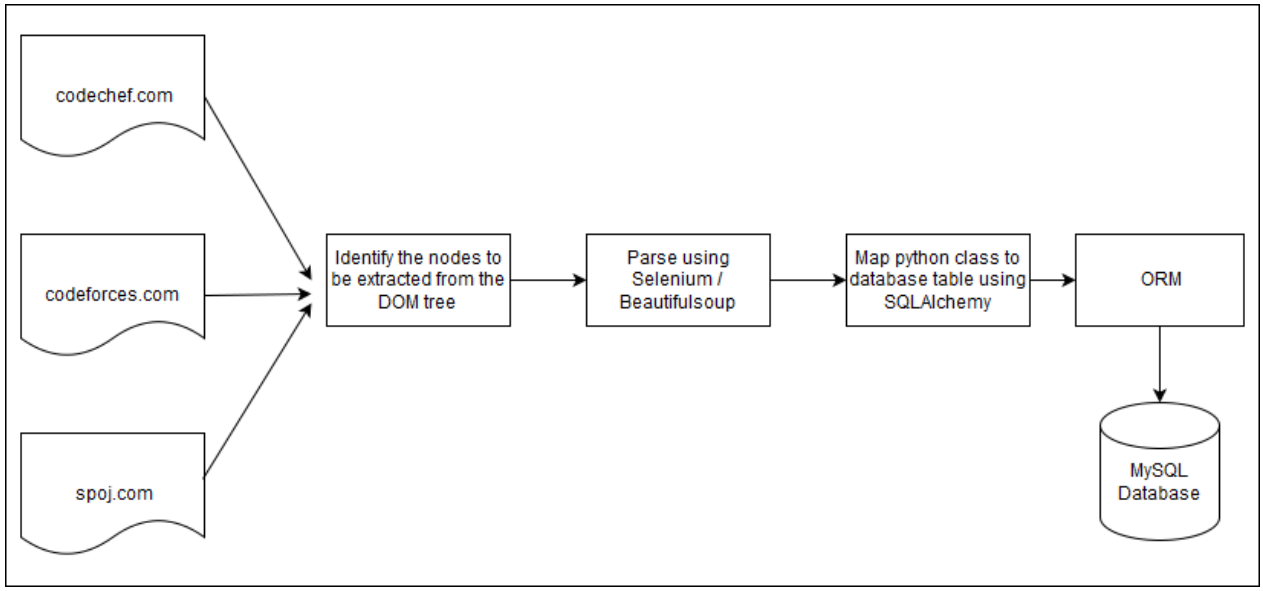


Figure 5.2: Extraction Process

Problems faced during extraction:

1. Certain DOM objects change dynamically and their references get lost
2. Scraping take long amount of time and any interrupt causes loss of the current processing
3. It is difficult to keep logs and track the progress
4. Dynamic changes to problems or users need to be re-scraped

5.1.3 Results

This section contains snapshots of database, for table User and table Problem respectively. Figure 5.3 shows first few rows from User table fetched from database. Figure 5.4 shows first few rows from Problem table fetched from database.

id	uname	country	city	is_student	pref_lang	rating_long	rating_short	rating_ltime	rank_long	rank_short	rank_ltime
80	bournecoder			True	c++	3950.61	1450.26	1516.6	7650 / 5984	4946 / 3573	2473 / 1852
81	mukel	Switzerland	Lausanne	True	c++	5874.21	2040.59	0	3871 / 3	1726 / 5	NA
82	ansh1star033	India	Gwalior	True	c++	5576.62	1000.52	638.06	4290 / 3299	12255 / 9462	16328 / 14143
83	king_of_hacker	India	Chennai	True	c++	6325.64	1378.62	1242.97	3308 / 2495	5703 / 4164	4991 / 3927
84	shahid_khan	India	Bengaluru	False	c	9479.74	2001.95	0	887 / 597	1843 / 1242	NA
85	revs	India	Jodhpur	True	c++	9463.31	1426.56	0	892 / 600	5161 / 3745	NA
86	raghu_317	India	Vijayawada	True	c++	11509.06	2243.63	1170.49	278 / 146	1254 / 796	6008 / 4794
87	hitesh091	India	New Delhi	True	c++	5261.91	1405.9	653.37	4814 / 3727	5385 / 3926	16131 / 13962
88	bhavesh_munot	India	Pune	False	python	5732.2	1551.22	1487.05	4077 / 3115	4023 / 2867	2673 / 2003
89	fauzdar65	India	Noida	False	c++	13859.12	3698.94	3218.55	63 / 18	120 / 30	50 / 14
90	atulshanbhag	India	Margao	True	c++	2005	1429.46	1403.21	16646 / 12961	5137 / 3724	3304 / 2517
91	arjunarul	India	Chennai	True	c++	3168.64	726.22	0	10119 / 7879	20450 / 16618	NA
92	anunay_arunav			True	java	3950.94	1182.43	1080.9	7646 / 5980	8486 / 6388	7479 / 6058

Figure 5.3: User Data Snapshot

id	prob_code	url	description	tag	submission_size	constraint	example_given	difficulty	category	time_limit
14	CHAHG	https://www.codechef.com/problems/CHAHG	Chef is an advocate for Go Green Initiative. Toda...	aug16 easy witalij_hq	3	Constraints 1 ? T ? 105 1 ? n ? 10 Subtask 1 (23 p...	True	easy	aug16 easy witalij_hq	2
15	TREECAKE	https://www.codechef.com/problems/TREECAKE	All submissions for this problem are available. Re...	kingofnumbers	3	Constraints 1 ? T ? 104 1 ? H ? 1018	True	easy	kingofnumbers	3
16	SECUBE	https://www.codechef.com/problems/SECUBE	All submissions for this problem are available. Re...	cook76 easy-medium iscsi modular-arith precomputa	100	Constraints 1 ? T ? 105 0 ? C ? K3 2 ? K ? 100	True	easy	cook76 easy-medium iscsi modular-arith precomputa	2
17	AVGSHORT	https://www.codechef.com/problems/AVGSHORT	All submissions for this problem are available. Re...	bellman-ford easy-medium graph ltime39 shortest-p	4	Constraints 1 ? N ? 500 1 ? M ? 1000 A is not equa...	True	easy	bellman-ford easy-medium graph ltime39 shortest-p	5

Figure 5.4: Problem Data Snapshot

5.2 Data Cleaning

When the text from the Competitive Coding Problem's Description is cleaned, it helps to increase the performance measure of the classifier. Cleaning process of the Problem Description is in three stages, as depicted in Figure 5.5

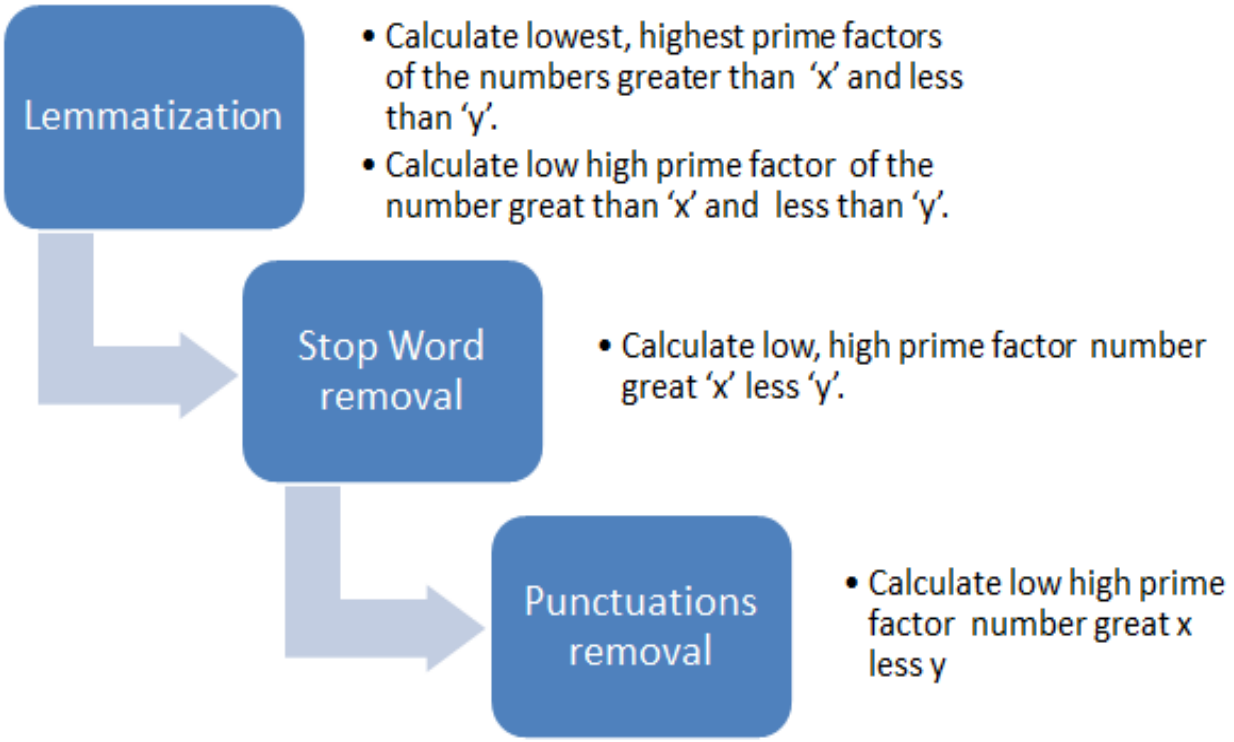


Figure 5.5: Stages of Cleaning Description

5.2.1 Problem Description Cleaning

In this section, we consider various methods to clean the problem description and those methods are as follows:

1. Lemmatization
2. Stop Word Removal
3. Remove Punctuation

a. Lemmatization and Stemming

For grammatical reasons, documents are going to use different forms of a word, such as organize, organizes, and organizing. Additionally, there are families of derivationally related words with similar meanings, such as democracy, democratic, and democratization. In many situations, it seems as if it would be useful for a search for one of these words to return documents that contain another word in the set.

The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. For instance: am, are, is be car, cars, car's, cars' car
The result of this mapping of text will be something like:
the boy's **cars** are different colors –> the boy **car** be differ color
However, the two words differ in their flavor.

Stemming usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes.

Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma.

If confronted with the token saw, stemming might return just s, whereas lemmatization would attempt to return either 'see' or 'saw' depending on whether the use of the token was as a verb or a noun. The two may also differ in that stemming most commonly collapses derivationally related words, whereas lemmatization commonly only collapses the different inflectional forms of a lemma. Linguistic processing for stemming or lemmatization is often done by an additional plug-in component to the indexing process, and a number of such components exist, both commercial and open-source.

b. Stop Word Removal

Stop words are the words we want to filter out before training the classifier. These are usually high frequency words that aren't giving any additional information to our labeling. In fact, they actually confuse our classifier. In English, they could be the, is, at, which, and on. The

important words we want to include when we're training text are the words that should be connected in some way to the classification label.
Consider following example,

"There are N servers which you have to place in n slots. Slots and servers are numbered from 1 to N. A distance between slots i and j is —i - j—. There are M pairs of servers that should be connected by wire. You are to place all the servers in the slots so the total wire length is minimized."

First, we need a list we can use in our code. A good node library we can use is stopwords - it's just an array of English stop words. Being rawText our unfiltered text, in the first line, we load an array of stop words using stopwords library. We then split the original text in words - also clearing out symbols and digits - and filter out the words in the stopwords array, and then convert everything back to a string.
Below is the result of stop word removal:

"There ~~are~~ N servers ~~which you have to~~ place in N slots. Slots ~~and~~ servers ~~are~~ numbered ~~from~~ 1 ~~to~~ N. A distance ~~between~~ slots i ~~and~~ j is —i - j—. There ~~are~~ M pairs ~~of~~ servers ~~that should be~~ connected ~~by~~ wire. You ~~are~~ to place ~~all the~~ servers ~~in the~~ slots ~~so the~~ total wire length ~~is~~ minimized."

Stop word elimination is a simple but it is an important aspect of many text mining applications as it has the following advantages:

1. Reduces memory overhead (since we eliminate words in consideration)
2. Reduces noise and false positives (since we are focusing on the more important terms)
3. Can potentially improve power of prediction (this is dependent on the application)

A point to note is that, while many applications benefit from the use of stop words, some do not see any added advantage in removing stop words other than the fact that it makes analysis or lookup much faster and reduces overall memory requirements.

c. Remove Punctuation

Punctuation remarks in the problem description don't play any role in pointing out the problem category. Such characters are for only English Grammar and human interpretation. Removing punctuation will help the supervised learning algorithm like SVM, K-nearest neighbors to improve further.

Consider following example of text for Lemmatization and Stemming, Initial Text: "There are N servers which you have to place in N slots. Slots and servers are numbered from 1 to N. A distance between slots i and j is —i - j—. There are M pairs of servers that should be connected by wire. You are to place all the servers in the slots so the total wire length is minimized."

After stop word removal:
"There ~~are~~ N servers ~~which you have to~~ place in N slots. Slots ~~and~~ servers ~~are~~ numbered ~~from~~ 1 ~~to~~ N. A distance ~~between~~ slots i ~~and~~ j is —i - j—. There ~~are~~ M pairs ~~of~~ servers ~~that should be~~ connected ~~by~~ wire. You ~~are~~ to place ~~all the~~ servers ~~in the~~ slots ~~so the~~ total wire length ~~is~~ minimized."

After Lemmatization, stop words removal and punctuation removal:
"There N server place n slot Slot server number 1 N A distance slot i j i j There M pair server connect wire You place server slot total wire length minimize."

5.2.2 Tags Cleaning

A tag of a competitive coding problem defines the category of the problem or simply the algorithm needed to solve it. A competitive coding problem typically has multiple tags. Since, most problems cannot be solved by any single method.
Tag contents:

1. User handles who had defined the problem
2. Algorithm for the problem
3. Level of difficulty of the problem

4. Contest where the problem occurred

Example: ad-hoc dp gcd jan17 medium wittyceaser
Tags like this are also not normalized.
For example:
tree : ['range-tree', 'kd-tree', 'link-cut-tree', 'fenwick-tree', 'tree', 'binary-tree', 'trees', 'segtree', 'segment-trees', 'tree-dp', 'suffix-trees', 'splay-tree', 'segment-tree']
A category like 'tree' has multiple tags like 'range-tree', 'binary-tree', or 'segtree'. So there are multiple words which define the same single category.

Solution:
To filter out such tags, we had created a Tag Map. The key of this map is a common prefix across all the dirty tags or a common term occuring across all the dirty tags and value is a set of those dirty tags.
math : 115
['math', 'maths', 'advanced-math', 'basic-math', 'simple-math', 'basic_math', 'basic_maths', 'mathematics']

dp : 106
['dpraveen', 'dp+lcs', 'dpmgc', 'dp', 'digit-dp', 'dp+bitmask', 'tree-dp']

ad : 88
['ad-hoc','adhoc']

greed : 64
['greedy-algo', 'greedychxorr', 'greedy']

tree : 60
['range-tree', 'kd-tree', 'link-cut-tree', 'fenwick-tree', 'tree', 'binary-tree', 'trees', 'segtree', 'segment-trees', 'tree-dp', 'suffix-trees', 'splay-tree', 'segment-tree']

grap : 51

['graph', 'grap', 'graphs', 'basic-graph', 'digraph', 'graph-theory']

dynamic : 44
['dynamic-bcc', 'dynamic-prog', 'dynamic']

The numbers are the counts of the problems where those dirty tags occured. We use the above Tag Map and match the key with the tag contents in the problem and then assign the standardize tag (e.g. greedy, dp, combinatorics, graph, math) for that problem. For example, as you can see in Figure 5.6, if in a problem tag 'segment-tree' comes, as it occurs as a dirty tag in the list of tags for key 'tree' we assign it our standardize tag 'tree'.

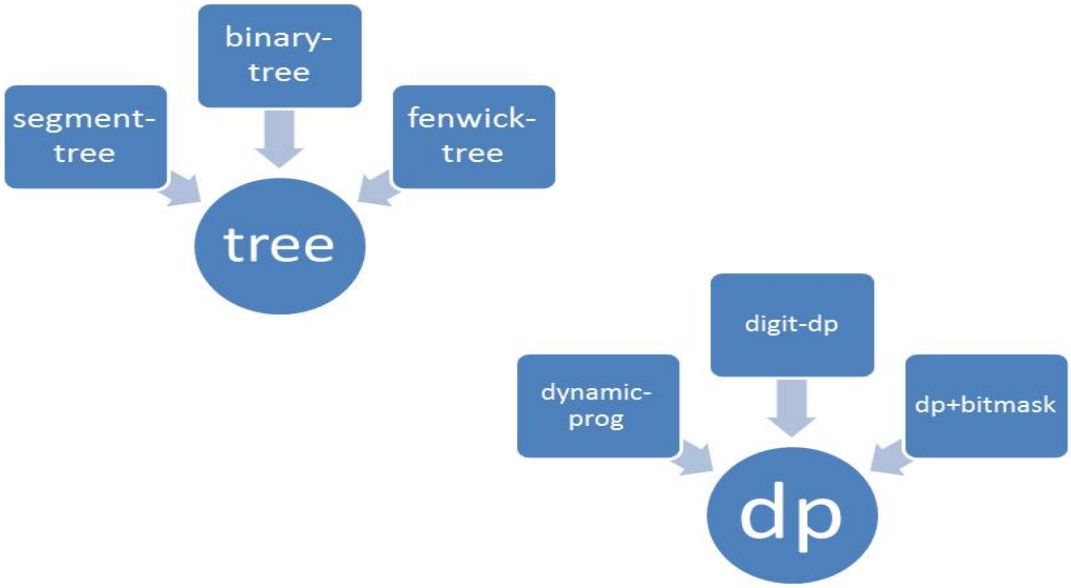


Figure 5.6: Tags Mapping

6 Classification

This chapter covers the primary aim of Competitive Problem Classification and Recommendation i.e. Classification or Tag prediction of problem into their algorithm classes. We study and implement different approaches. At the end of this current stage we perform parameter tuning to improvise over our model.

6.1 Model Training

Two approaches were identified for tag prediction for the problems, one was to build a global multiclass classifier and then use method such as One vs All to select the final class. The second approach was to build a discriminant classifier for each of the tags and then predict the final tags choosing the most likely tags. Since we need to predict more than one tags for each of the probelm, it was decided to use the first approach. So that we can find out top k tags classifier which gave highest score for that problem.

6.1.1 Frequent Pattern Mining

In this section we consider frequent pattern mining approach to generate features. Also, one-vs-rest model is consider for dataset generation. The one-vs.-rest (or one-vs.-all, OvA or OvR, one-against-all, OAA) strategy involves training a single classifier per class, with the samples of that class as positive samples and all other samples as negatives. This strategy requires the base classifiers to produce a real-valued confidence score for its decision, rather than just a class label; discrete class labels alone can lead to ambiguities, where multiple classes are predicted for a single sample.

a. Feature Selection

Once data is well cleaned and integrated in the database, we can start the process of training the model. In that we first fetch all problems from database using SQL Alchemy which is ORM API for python and MYSQL Database. We then process each problem individually. For each such problem we first check if it is of category for which we are building the model. If it is then for each new word in the problem we insert it in a dictionary and if the word is already present in dictionary we increment its count. Similarly separate dictionary is created for the words in the problems which belong to different category.

Table 6.1.1 depicts counts of words while considering graph as category. First column indicates the word considered, Second indicates number of graph problems in which that word occurred for at least once. Third indicates number of non-graph problems in which that word occurred for at least once. Next column contains the total numbers of problems in which that word occurred. And last column states percentage occurrence of word in graph problems. Here we consider only those words which have occurred in at least 50 problems.

This table allows us to have formal representation of words and there occurrence. Once we are ready with the table, then we sort it according to the percentage in non-increasing order. Using this, we learn the relative importance of the words and we can use this information for deciding our feature words. So for selecting words as feature we take top 10 words from the table and bottom 10 words from table. A sample set of top 10 words and bottom 10 words for graph problems are shown in Table 6.1.1 and Table 6.1.1. Reason for doing so is that, top 10 words signify that these words are most likely to be present in graph problems, while bottom 10 words signify that these words are least likely to be present in graph problems. In this way we choose our features.

Word	Count in graph problems	Count in non-graph problems	Total Count	Percentage
determine	37	18	55	67.2727
graph	93	48	141	65.9574
undirected	33	18	51	64.7058
city	66	36	102	64.7058
...	...	...	...	...
remainder	2	68	70	2.8571
divisor	2	71	73	2.7397
palindrome	1	50	51	1.9607
interval	1	55	56	1.7857

Table 6.1: Sample distribution of feature frequencies for graph category

determine	graph	city	undirected	bidirected	decide	memoize	megabyte	choose	wa
67.27	65.96	64.71	64.71	62.07	62.03	59.65	59.47	59.26	59.02

Table 6.2: Top 10 words for graph category

nth	circle	12	math	remainder	divisor	palindrome	interval	1ine	ordered
3.17	2.99	2.99	2.90	2.86	2.74	1.96	1.79	1.72	0

Table 6.3: Bottom 10 words for graph category

b. Data Set Creation

When we are done with selection of features then we can use following algorithm to generate the dataset, in the format suitable for model training. Following list is created for each problem:

[feature1, feature2, feature3, ...,feature20, class]

where:

feature1, feature2 are the feature words that we choose from above process,

class is whether that problem belongs to the category for which we are building the dataset.(i.e 1 if problem is of 'dp' category and 0 otherwise).

Now for each word from problems description we check if that word is in feature set, if it is then value in the corresponding index of list is set to 1, once this process is finished remaining values are filled with 0. This process is repeated for each problem from problem collection. The data set created this way is feed to classifier and model is created for each category of problems. Algorithm 6.1 is the complete algorithm for creating features from

Step 1: Create two dictionaries dic0 and dic1

Step 2: for each word W in the problem
if problem is of chosen category
if W is not in dic1
add W to dic1
else increment count of W in dic1
else
if W is not in dic0
add W to dic0
else increment count of W in dic0
Repeat step 2 for each problem

Step 3: Create table containing count of each word in dic0,dic1.
Calculate total-count (dic0+dic1) and percentage (100*dic1)/total-count
Remove the words with total-count¡50

Step 4: Sort the words in non-increasing order of percentage
Add top 10 and bottom 10 words to feature list.

Step 5: Create feature list for each category using above steps

Figure 6.1: Feature Creation Algorithm

problme desctiption. Algorithm 6.2 is the set of steps for creating the dataset for perticular category given features from algorithm 6.1

6.1.2 TF-IDF

tf-idf is a model which guides to decide the features of the documents. It is mainly use for machine learning problems involving text. Problems where an agent needs to label a document based on words inside the document. In such cases, how important a word is for a document? how a particular word is associated with the document?, questions arises. We may try to use the 'direct count' of words to associate them with the document. But in some cases, it may turn out not much fruitful.

For example, we have set of problem descriptions. And we have a set of key word {'find', 'length', 'optimal'} which occurs more in a problem having 'greedy' Category. So if a word like 'optimal' and 'chef' occurs more number of time for a problem, then we may resolve it to 'greedy' category. But in reality that problem may belong to category of 'graph' problems because there are problem descriptions like "find optimal length/path" in 'graph'

Step 1: Declare list L{feature1:0, feature2:0,....feature20:0, class:0} for chosen category

Step 2: for each word W in the problem
if W one of the features in L
mark the feature with value 1
if problem belongs to chose category
Set class to 1
Repeat step 2 for each problem

Step 3: Repeat above process for each category

Figure 6.2: Dataset Generation Algorithm

problems. So, our prediction will skew from 'graph' to 'greedy' problem. The important word in such case is 'length'. When word like 'length' occurs in conjunction with 'optimal' enough number of times then the prediction algorithm will find easy to resolve to category 'graph'. This happens, because 'optimal' and 'chef' occur in almost every problem, while words like 'length' occur in rare problems. Hence, to reduce the effect of words which occur globally more number of times than a set of conjunctive words which occur less number of times but are more relevant to the document, we have to use tf-idf.

Tf-idf incorporates local and global occurence influences, it considers not only the isolated term in a document but also the term within document collection. It scales down the frequent terms and scales up the rare terms that matter for that particular class.

Calculation of tf-idf

$$\mathrm{tf}(t,d) = \sum_{x \in d} \mathrm{fr}(x,t)$$

where the $\mathrm{fr}(x,t)$ is a simple function defined as:

$$\mathrm{fr}(x,t) = \begin{cases} 1, & \text{if } x = t \\ 0, & \text{otherwise} \end{cases}$$

'd' refers the document in the data set
't' refers to the term

$$\mathrm{idf}(t) = \log \frac{|D|}{1 + |\{d : t \in d\}|}$$

'D' refers to the document set or our data set

$$\text{tf-idf}(t) = \mathrm{tf}(t,d) \times \mathrm{idf}(t)$$

6.1.3 Random Forest Training

We use Random Forest classification to evaluate the modeling strategy. For classification purpose, we randomly split the reconstructed data set into training data set and test data set, then use scikit-learn package to train a binary classification RF model namely to make trading decisions, which is either make a bet 1 or do nothing 0

6.2 Result Analysis

We consider following performance measures for machine learning models from Section 3.2 on Problem Classification.

- Precision - How many problem's classes were correctly predicted out of total classes predicted?
- Recall - How many problem's classes were correctly predicted out of total correct classes?
- F measure - It is based on combination of weighted Precision and Recall

6.2.1 Testing and accuracy, precision, recall calculation

We apply the trained model to test data set, and obtain the following results viz. accuracy, precision and recall for each pair of stocks. Recall is the vital parameter of tag prediction of competitive coding problem. Our model utilizes F measure which is weighted measure of both Precision and Recall.

- Accuracy : The accuracy is the proportion of the total number of predictions that were correct.

$$Accuracy = \frac{true\ positive + true\ negative}{true\ positive + false\ positive + true\ negative + false\ negative}$$

- Precision : The precision is the proportion of the predicted positive cases that were correct.

$$Precision = \frac{true\ positive}{true\ positive + false\ positive}$$

- Recall : The recall is the proportion of the total positive cases that were found.

$$Recall = \frac{true\ positive}{true\ positive + false\ negative}$$

You can observe, from Table 6.4, that maximum fscore was achieved for combinatorics and minimum fscore was achieved for math. This was because alpha (false positive + false negative) out of total problems was more for math and least for combinatorics. This helps us to analyze advantages or any improvements in current model.

Category	Precision	Recall	FScore
greedy	0.786997	0.938776	0.856212
graph	0.912295	0.986702	0.948041
tree	0.949838	0.994073	0.971452
combinatorics	0.96108	0.999174	0.979757
math	0.800893	0.897	0.846226
dp	0.780508	0.95243	0.857941

Table 6.4: Results for Codeforces: Frequent Pattern Mining

A sample graphical visualization of performance results by Frequent Pattern Mining technique for classification a class positively is shown in given Figure 6.3. Also, performance results by Frequent Pattern Mining technique for classification a class negatively is shown in given Figure 6.4

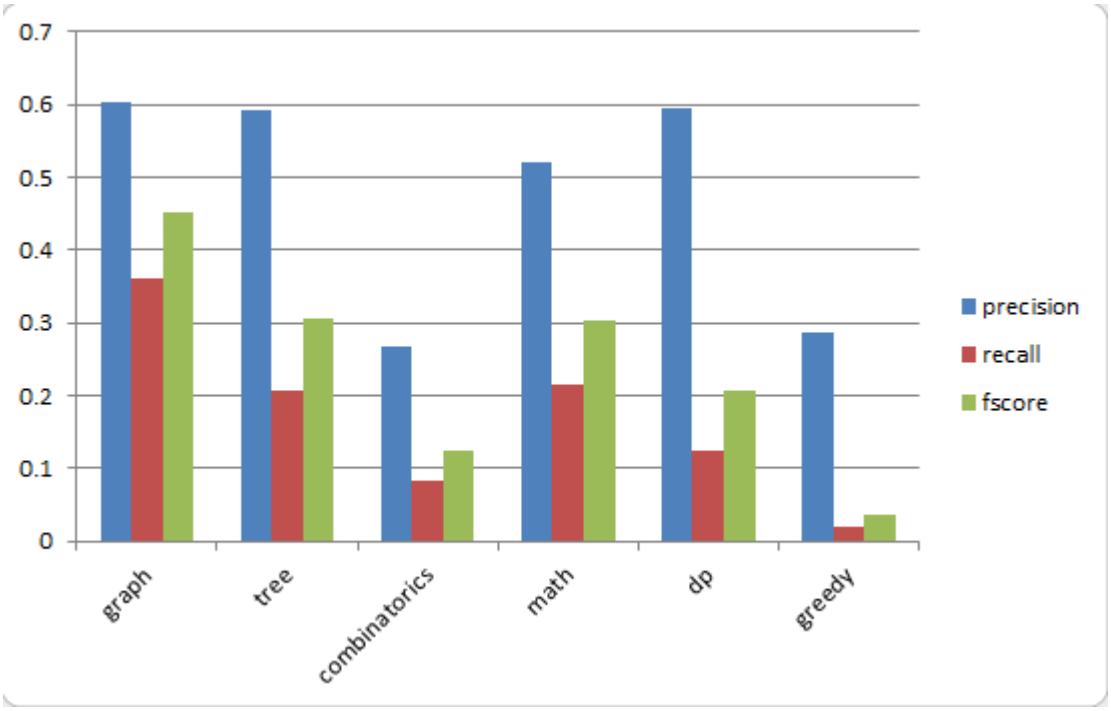


Figure 6.3: Positive results using Frequent Pattern Mining

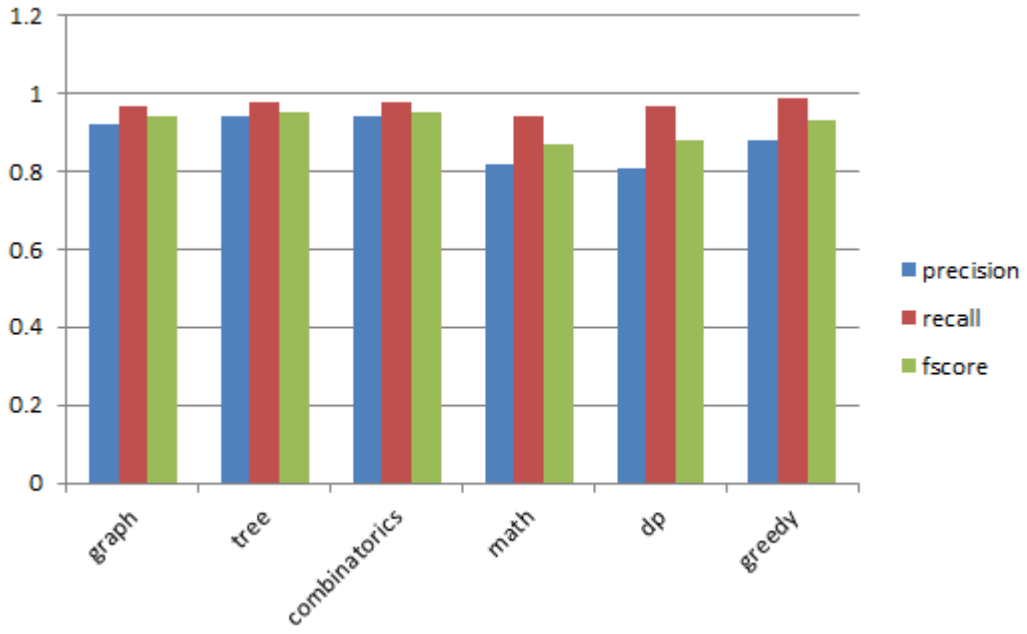
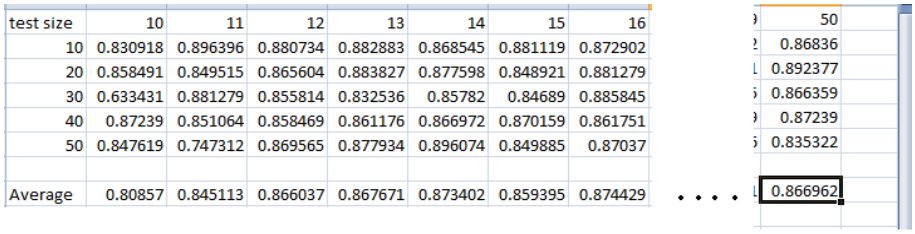


Figure 6.4: Negative results using Frequent Pattern Mining

6.2.2 Parameter Tuning

From frequent pattern mining result, we can use top 'x' amount of words and bottom 'x' amount of words as features. Also, while training the data, we partition data into test set and



test size	10	11	12	13	14	15	16	...	50
10	0.830918	0.896396	0.880734	0.882883	0.868545	0.881119	0.872902		0.86836
20	0.858491	0.849515	0.865604	0.883827	0.877598	0.848921	0.881279		0.892377
30	0.633431	0.881279	0.855814	0.832536	0.85782	0.84689	0.885845		0.866359
40	0.87239	0.851064	0.858469	0.861176	0.866972	0.870159	0.861751		0.87239
50	0.847619	0.747312	0.869565	0.877934	0.896074	0.849885	0.87037		0.835322
Average	0.80857	0.845113	0.866037	0.867671	0.873402	0.859395	0.874429	....	0.866962

Figure 6.5: F1 Score by Test Size and Words set

train set. The wall of partition, 'y', can be 90-10, 80-20, 70-30 and so on. Notice, there isn't any thumb rule to fix value of 'x' and 'y' at such places where we can get possible results. To investigate value of 'x' and 'y' where we can find best possible result, we conducted following experiment.

We started our wall partition from 90-10, that is 10 percent test size and 90 percent train set. And for each such possible wall, we train data for values of 'x' in range from 10 to 50.

As you can observe from Figure 6.5 for 'x' as 16, we found best possible f score of 87 percent over all average of 'y'. Hence, it suggests for any value of 'y' if we took top 16 words and bottom 16 words we have greater chance to achieve good results.

7 Recommendation System

This chapter covers secondary aim of Competitive Problem Classification and Recommendation i.e. Recommendation Engine. In majority of time users spending time on Competitive Coding platforms like Codechef, Spoj and Codeforces don't have much intution to decide which problems to attempt. Such users either keep on solving easier problems or they try to solve harder problems expectation that it will gurantee their improvement. Users don't consider which category of problems they must try to attempt to solve such that they get to learn an unknown algorithm concept. Considering such issues the goal of Recommendation Engine is:

1. To recommend problems that the user must be able to solve or in other words problems enusuring higher likelihood to be solved

2. To recommend problems with such degree of difficulty higher than their current level, also not much higher that they won't be able to solve

3. To adapt recommendations so that a user gradually progresses than their current level to higher level in each category

In this chapter we consider different approaches to recommend problems. We implement one of the approach to satisfy the first requirement of the recommendation engine (ensuring higher likelihood to be solved). Also we add a layer on recommendations that we get from engine to guarantee the second and third requirement.

7.1 Approaches

1. **Content based algorithms:**

   If you like an item then you will also like a "similar" item. Based on similarity of

the items being recommended. It generally works well when its easy to determine the context/properties of each item. For instance when we are recommending the same kind of item like a movie recommendation or song recommendation.

2. **Collaborative filtering algorithms:**

   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. This algorithm is entirely based on the past behavior and not on the context. This makes it one of the most commonly used algorithm as it is not dependent on any additional information. For instance: product recommendations by e-commerce player like Amazon and merchant recommendations by banks like American Express. Further, there are several types of collaborative filtering algorithms:

   (a) **User-User Collaborative filtering:**

   Here we find look alike customers (based on similarity) and offer products which first customer's look alike has chosen in past. This algorithm is very effective but takes a lot of time and resources. It requires to compute every customer pair information which takes time. Therefore, for big base platforms, this algorithm is hard to implement without a very strong parallelizable system.

   (b) **Item-Item Collaborative filtering:**

   It is quite similar to previous algorithm, but instead of finding customer look alike, we try finding item look alike. Once we have item look alike matrix, we can easily recommend alike items to customer who have purchased any item from the store. This algorithm is far less resource consuming than user-user collaborative filtering. Hence, for a new customer the algorithm takes far lesser time than user-user collaborate as we don't need all similarity scores between customers. And with fixed number of products, product-product look alike matrix is fixed over time.

7.2 Word2Vec

In this section we explore how word2vec conforms to the need of Recommendation Engine and its operations.

7.2.1 Reason to use word2vec

a. Problems in Content Based Filtering

Users can only receive recommendations similar to their earlier experiences. In our context: A user would get only same category problems of same level, which they had solved in the past.

b. Problems in Collaborative filtering

Gray sheep refers to the users whose opinions do not consistently agree or disagree with any group of people and thus do not benefit from collaborative filtering. In our context: There are users which solve few problems in each category-level combination. They do not fit into any user group.

c. Solution provided by word2vec

In contrast to Content Based Filtering, word2vec recommends problems, not only considering the user's past experience but also other user's past experience. Hence, it does not suffer deficiencies of CBF. Word2Vec does not suffer from gray sheep problem. As it does not generalize on utility matrix (whether a particular user solved a particular problem or not). Because word2vec recommends problems based on user problem solving patterns in time and not on whether a particular user-problem solved matrix. It considers time factor in pattern learning.

7.2.2 Operation of Word2Vec

This section contains covers the functioning within word2vec. It explains word vector representation and two methodologies for word2vec: CBOW and Skipgram.

a. Word vector

At one level, it's simply a vector of weights. In a simple 1-of-N (or 'one-hot') encoding every element in the vector is associated with a word in the vocabulary. The encoding of a given word is simply the vector in which the corresponding element is set to one, and all other elements are zero. Suppose our vocabulary has only five words: Soviet, Union, Russia, America and Taiwan. We could encode the word 'Union' as shown in Figure 7.1

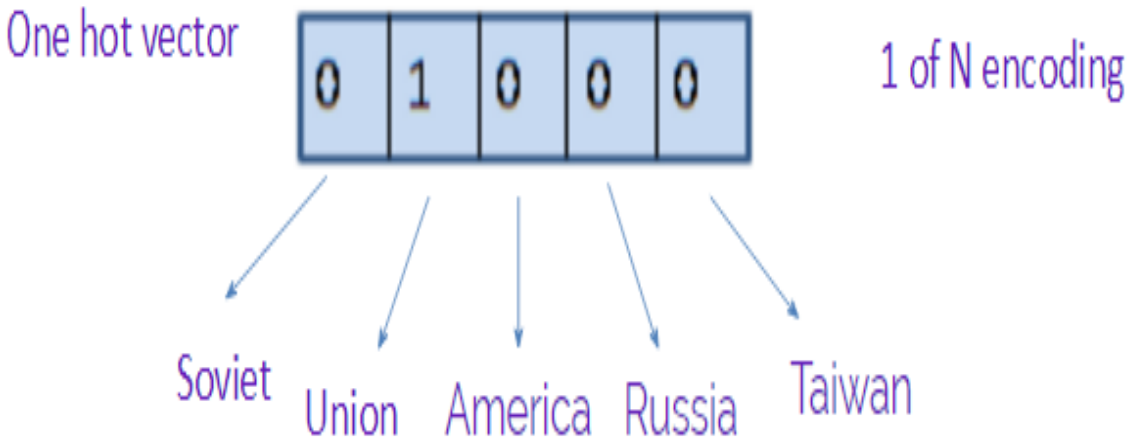


Figure 7.1: One of N Encoding

Using such an encoding, there's no meaningful comparison we can make between word vectors other than equality testing.

In word2vec, a distributed representation of a word is used. Take a vector with several hundred dimensions (say 1000). Each word is represented by a distribution of weights across those elements. So instead of a one-to-one mapping between an element in the vector and a word, the representation of a word is spread across all of the elements in the vector, and each element in the vector contributes to the definition of many words.

Such a vector comes to represent in some abstract way the 'meaning' of a word. And as we'll see next, simply by examining a large corpus it's possible to learn word vectors that are able to capture the relationships between words in a surprisingly expressive way. We can also use the vectors as inputs to a neural network.

We find that the learned word representations in fact capture meaningful syntactic and semantic regularities in a very simple way. Specifically, the regularities are observed as constant vector offsets between pairs of words sharing a particular relationship.

Somewhat surprisingly, it was found that similarity of word representations goes beyond simple syntatic regularities. Using a word offset technique where simple algebraic opera-

tions are performed on the word vectors, it was shown for example that vector("Union") + vector("Russia") results in a vector that is closest to the vector representation of the word "Soviet". This is depicted in Figure 7.2

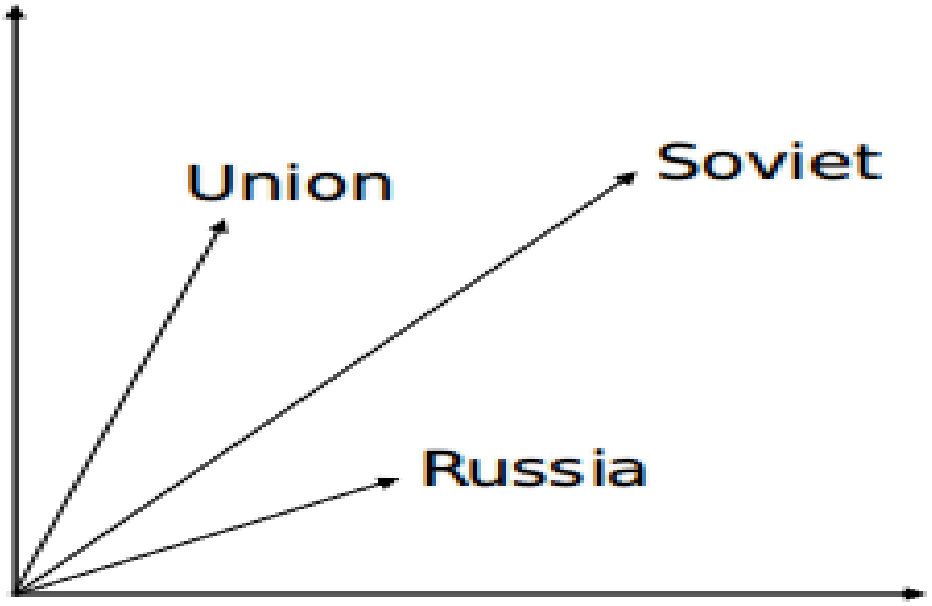


Figure 7.2: Word Vectors Graph

b. Approaches

Word2vec can utilize either of two model architectures to produce a distributed representation of words: continuous bag-of-words (CBOW) or continuous skip-gram. In the continuous bag-of-words architecture, the model predicts the current word from a window of surrounding context words. The order of context words does not influence prediction (bag-of-words assumption). In the continuous skip-gram architecture, the model uses the current word to predict the surrounding window of context words. The skip-gram architecture weighs nearby context words more heavily than more distant context words. According to the authors' note, CBOW is faster while skip-gram is slower but does a better job for infrequent words. Example,

World-War 2 was started by **Germany**, and Nazi faction was the main reason.

So, whenever Germany word comes, World-War 2 and Nazi must be returned by skipgram

Consider a piece of prose such as ”The recently introduced continuous Skip-gram model is an efficient method for learning high-quality distributed vector representations that capture a large number of precises syntatic and semantic word relationships.” Imagine a sliding window over the text, that includes the central word currently in focus, together with the four words taht precede it, and the four words that follow it.

It is constructed with the focus word as the single input vector, and the target context words are now at the output layer. This is depticted in Figure 7.3

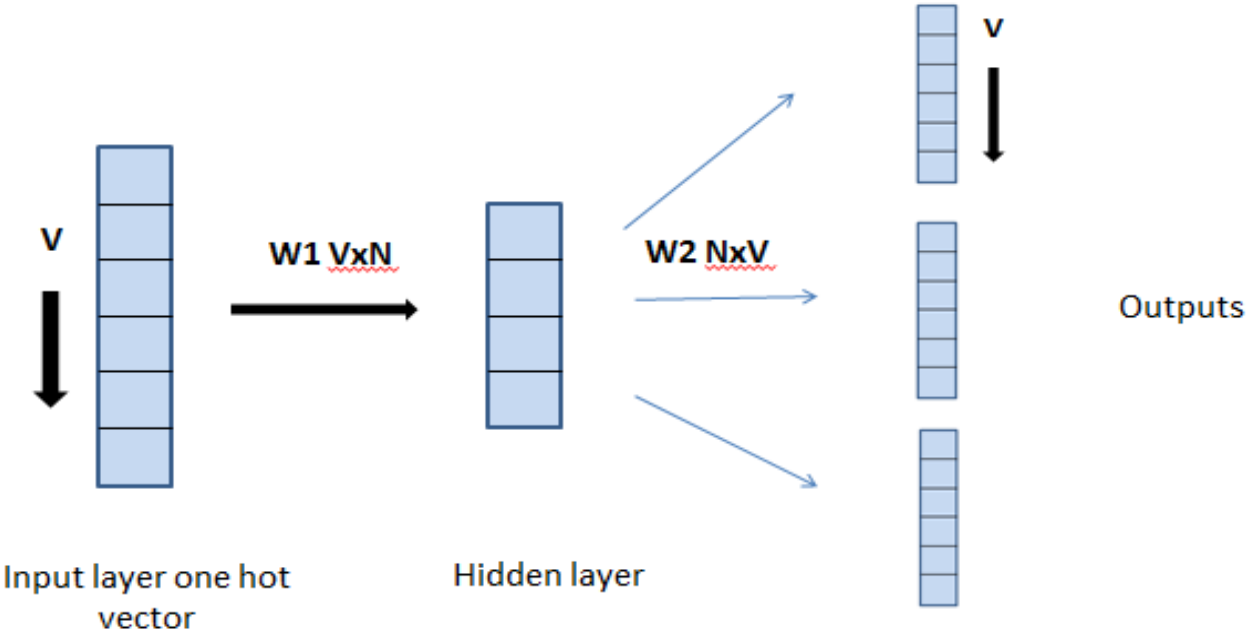


Figure 7.3: Skipgram Example

The activation function for the hidden layer simply amounts to copying the corresponding row from the weights matrix W1 (linear) as we saw before. At the output layer, we now output C multinomial distributions instead of just one. The training objective is to mimimize the summed prediction error across all context words in the output layer. In our example, the input would be ”learning”, and we hope to see (”an”, ”efficient”, ”method”, ”for”, ”high”, ”quality”, ”distributed”, ”vector”) at the output layer.

c. Reason to use skipgram

The main reason is that the CBOW smooths over a lot of the distributional statistics by averaging over all context words while the skipgram does not. With little data, this "regularizing" effect of the CBOW turns out to be helpful, but since data is the ultimate regularizer the skipgram is able to extract more information when more data is available. Skip-gram: works well with small amount of the training data, represents well even rare words or phrases. CBOW: several times faster to train than the skip-gram, slightly better accuracy for the frequent words.

7.3 Improvement Factor

As we conveyed in the introduction of this chapter that second and third goals ensure improvement of the user. We need some strategy to decide user level and recommend problems according to it. We need to ensure growth in user's ability. In this section we explain, how to achieve it in our recommendation system.

At first, we need to identify user level, so as to adapt the recommendation list for that specific user. We need to identify those problems from recommendation list, which will be helpful for users. For example, a user solving easy problems under math category for a certain amount of time, must receive medium level problem under math category. Also, we cannot keep on recommending problems to user on the same level. To ensure improvement we have to determine user level for each category and then decide what next problem level should be in that category. Using this approach, we will promote user gradually from easiest level to hard. This similar approach has been implemented by InterviewBit platform (A platform for coding interview questions).

a. Determining User Level for each category

We use following statistical approach to determine how many problems at particular level a user should solve to get promoted to next level. Also, this result will be different for each category. We maintain two list of tipping points for each category: one for Easy to Medium

and one for Medium to Hard. Easy-To-Medium: Each member in this list resembles how many problems to solve in the Easy level to get medium level. Medium-To-Hard: Each member in this list resembles how many problems to solve in the Medium level to get Hard level.

b. Steps to build tipping point list:

1. For each user, group problem submissions by category
2. For each category sort the problems by time, create a level sentence (a sentence containing only level values sorted by time)
3. For each medium level value in this level sentence, we insert count of easy level values before it inside Easy-To-Medium list
4. For each hard level value in this level sentence, we insert count of medium level values before it inside Medium-To-Hard list.
5. Apply max function on the list to decide how many problems at particular level a user should solve to get promoted to next level

The above list represents tipping point or effort needed to overcome and promote to next level. To generalize over values in this list, we tried different statistical functions like max, median, mean, mode. We use max instead of mean/median/mode because using max, we assure that a user concretely has achieved previous level before he/she gets promoted to next level.

7.4 Steps for Recommendation

As, we had chosen word2vec for recommendation system model, we formalise the steps for building the recommendation system as shown in Figure 7.4. Steps given in Figure 7.4 is sequence of actions that are followed to get the problems, that will ensure the improvement in problem solving skills of user.

1. Train the word2vec model
   (a) Sort submissions for user by difficulty then sort each difficulty by date
   (b) Generate submission sentence for each user containing words (which is problem code based on sorting
   (c) Train the model by sentences with parameter tuning. Window for distance comparing between words (problem codes skip-gram specification)
2. (a) Get user levels by category based on tipping point
   (b) Build category wise limits based on tipping points
   (c) For a novice user recommend popular problems (this is likely case due to cold start)
   (d) For a regular user recommend problems based on the model

Figure 7.4: Recommendation Steps

7.5 Result Analysis

For anaylising performance of our recommendation engine, as we split the data for train-test chunks, we splitted the data for the user by removing last k problems. The last k submissions become part of validation set while n - k submissions become part of training set.

We calculate precision and recall based on following parameters,
tp := True positive, the number of problems the model recommended and was solved by user in future time (as it was part of validation set)
fp := False positive, the number of problems the model recommended and was not solved by user in future time (as it was not part of validation set)
fn := False negative, the number of problems the model not recommended but was solved by user in future time (as it was part of validation set)
tn := True negative, the number of problems the model not recommended and was not solved by user in future time (as it was not part of validation set)

Following, Table 7.1, is the confusion matrix for above variables, here columns are the problems recommended by our model, and rows are the problems that user actually solved.

	Predicted Solved	Predicted Not Solved
Actual Solved	$tp = 2892$	$fn = 7708$
Actual Not Solved	$fp = 13308$	$tn = x$

Table 7.1: Confusion Matrix for Recommendation Engine

True Negative does not have any significance in this model because tn suggests that user has not solved unrecommended problems which is irrelevant.
Following values are calculated based on the formulae explained in Section 6.2
Precision: = 18.18
Recall := 27.28
F1Score := 21.82

We had calculated above values by fine tuning parameters like Validation Set Size, Window Size and Number of Recommendation.
The above performance measure are calculated without feedback. Validation is simply performed by splitting the data point in time. This performace measure values represent the likeliness of user to solve the problem with their own choice and not considering our engine. The real environment based performace measure will consider feedback loop where we will compare behaviour of the user after using our recommendation engine.

8 Summary and Conclusion

At the end of our research and project, we had summarize and revisit our findings in concise.

8.1 Summary/Learnings

During the development of this project, we understood how competitve programming is important, how it influences the IT industry and Computer Science Graduates. Competitve Programming is one of the way Universities make their mark in academia. Competitve Programming reflects the talent present at Universities. Through Competitive Programming students build their problem solving skills which is essential in day to day task at tech-giant product based companies (Facebook, Twiteer, Google). It is also important for rigorous Coding Interviews carried out by this tech-giants.

Currently, India ranks a lot behind for ACM-ICPC organised annually. Hardly, one team gets selected for Finals from India or in rare cases none. India is becoming IT Hub but this Hub bubble contains in majority only Service Based Companies. Although few tech giant offices are present but very few Computer Science or IT graduates work at such places. Improvement of Computer Science/IT Graduates in Competitive Programming is necessary.

Different competitve programmer use different strategies to improvise their skills. They find out their own shortcomings manually in different categories and solve problems in that category. Such strategies are manual and are based on human intution and hypothesis. Here, we hopped into machine learning for predicting classes of problems and also to build a Recommendation Engine through Data Mining.

In this project we learnt about different classification models. In the process, we also learn about statistical Natural Language Processing techniques like Frequent Pattern Timing and Term Frequency-Inverse Document Frequency technique.

We learn how a Recommendation Engine using Content Based Filtering - Collaborative

Filtering works. And at last, we learn about word2vector NLP model and adapted it for our recommendation system.

Our end product for Recommendation is a Web Page implemented through Flask (for back-end) and Javascript Front End. For tag prediction, we had implemented Google Chrome Extension.

8.2 Conclusion

1. For classification KNN performed almost equally to Random Forest. On overall average scenarios Random Forest performed better than KNN. Also Random Forest does not require any domain knowledge (like number of nearest neighbours to inspect), we used Random Forest for classification

2. As explained in section 7.5 (Result Analysis section of Chapter Recommendation System) we made an estimate of our performance by splitting data into train-test set by time attribute and evaluated accuracy through standard Confusion Matrix methodology. Although results show that our accuracy is less. This result does not reflect the real world influence. An actual estimate will be based upon the feedback loop where we evaluate the system based on whether user was able to solve the problem recommended and whether there was any growth in the user. Therefore, we consider building a real world feedback loop.

References

[1] Competitive-Programming:https://en.wikipedia.org/wiki/Competitive_programming

[2] Codechef https://www.codechef.com/ It is a non-profit educational initiative of Directi for competitive programming

[3] Spoj http://www.spoj.com/ Sphere Online Judge is an online judge system for competitive programming with over 315,000 registered users and over 20,000 problems

[4] Codeforces http://codeforces.com/ It is a Russian website dedicated to competitive programming

[5] HackerEarth https://www.hackerearth.com/ It is a recruitment and competitive coding platform

[6] HackerRank http://hackerrank.com/ It is a technology company that focuses on competitive programming challenges for both consumers and businesses

[7] Anudeep Nekkanti https://blog.anudeep2011.com/machine-learning-everywhere-why-not-in-competitive-programming/ Machine Learning in Competitive Programming

[8] C. Apte, F. Damerau, and S. M. Weiss. *Automatic learning of decision rules for text categorization* Research Report RC19979(82518), IBM, 1993.

[9] Tokunaga, Takenobu, Iwayama, Makoto1. *Text categorization based on weighted inverse document frequency. March 1994*

[10] H. P. Luhn. *A statistical approach to mechanized encoding and searching of literary information* IBM Journal of Research and Development, Vol. 1, No. 4, pp. 307{319, 1957}

[11] K. Sparck Jones. *A statistical interpretation of term specificity and its application in retrieval* Journal of Documentation, Vol. 28, No. 1, pp. 11{21, 1972}.

[12] Huang Chuangguang, Yin Jian, Wang Jing, et al.*Collaborative filtering algorithm of uncertain neighbor [J]* Journal of Computer Science, 2010, (8):1369-1377.

[13] Yoav Goldberg, Omer Levy. *Word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method[J]* Eprint Arxiv, 2014.

[14] Clayton Stanley (clayton.stanley@rice.edu) Department of Psychology, Michael D. Byrne (byrne@rice.edu) *Predicting Tags for StackOverflow Posts* Departments of Psychology and Computer Science, Rice University.

[15] Sebastian Schuster Wanying Zhu. fsebschu,wanyingz,yiychengg@stanford.edu, Yiying Cheng *Predicting Tags for StackOverflow Questions*

[16] Chintan Parikh, chintanp@stanford.edu*Identifying Tags from millions of text question,*

[17] Hicham Hage, Esma Aimeur *Exam Question Recommender System* Department of CS, Montreal University

[18] Jiaji Hu, Xuening Liu, Li Yi *Keyword Extraction for Stack Exchange Questions*

[19] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean *Efficient Estimation of Word Representations in Vector Space*

[20] Yao Xiao, Quan Shi *Research and Implementation of hybrid recommendation Algorithm* Nantong University

[21] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean *Distributed Representations of Words and Phrases and their Compositionality* Google Inc. Mountain View

[22] Mikolov,Tomas *Efficient Estimation of Word Representations in Vector Space* arXiv:1301.3781

[23] Classification using k-Nearest Neighbors in R. http://en.proft.me/2017/01/22/classification-using-k-nearest-neighbors-r/

[24] Introduction to Support Vector Machines. http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html

[25] Neural Network. https://i.stack.imgur.com/1bCQl.png

[26] Single Neuron Neural Network. https://i.stack.imgur.com/E6SsE.png

[27] Collaborative Filtering — Stanford University. https://www.youtube.com/watch?v=h9gpufJFF-0

[28] Precision and recall. https://en.wikipedia.org/wiki/Precision_and_recall

Acknowledgements

The success and final outcome of this project required a lot of guidance and assistance from many people and we are extremely fortunate to have got this all along the completion of our project. Whatever we have done is only due to such guidance and assistance and we would not forget to thank them. We especially thank Dr. S.G.Bhirud for giving us the support, guidance and some very useful insights which helped us complete the work in time.

Date: May 5, 2017

Veermata Jijabai Technological Institute