### SELF LEARNING BINARY TREE FOR JAVA OBJECTS

#### A PROJECT REPORT

***Submitted by***

#### Abhishek M

(16030141CES006)

**Jay Dev Rai M** (16030141CSE037)

**Meghana S**  (16030141CSE054)

**Naveen Kumar J R** (16030141CSE059)

*In partial fulfilment for the award of the degree of*

### BACHELOR OF TECHNOLOGY IN

**COMPUTER SCIENCE AND ENGINEERING**

Under the guidance of

### Prof. R. VijayaLakhsmi Asst. Professor



**Department of Computer Science and Engineering**

**ALLIANCE COLLEGE OF ENGINEERING AND DESIGN**

Chikkahagade Cross Chandapura-Anekal Main Road Bangalore-562106

#### MAY- 2020

**Department of Computer Science and Engineering**

Alliance College of Engineering and Design, Alliance University

Chikkahagade Cross Chandapura-Anekal Main Road, **Bangalore-562106**



***CERTIFICATE***

This is to certify that the project work entitled **“Self Learning Binary Tree For Java Objects”** is a bonafide record for work done by **Abhishek M** (16030141CSE006), **Jay Dev Rai M**(16030141CSE037), **Meghana S** (16030141CSE054) and **Naveen Kumar J R**(16030141CSE059), submitted in partial fulfillment of the requirements of the award of the degree **Bachelor of Technology** in **Computer Science and Engineering** during the year 2019-2020.

#### Prof. Vijaya Lakshmi R Dr. Shekhar. R

Project Guide Head of the Department



***Declaration***

This is to declare that the report titled “**Self Learning Binary Tree For Java Objects**” has been made for the partial fulfilment of the Course Bachelor of Technology in Computer Science and Engineering, under the guidance of **Prof. Vijaya Lakshmi R.**

I confirm that this report truly represents my work undertaken as a part of my project work. This work is not a replication of work done previously by any other person. I also confirm that the contents of the report and the views contained therein have been discussed and deliberated with the faculty guide.

|  |  |  |
| --- | --- | --- |
| **NAME** | **REG. NO** | **SIGNATURE** |
| Abhishek M | 16030141CSE006 |  |
| Jay Dev Rai M | 16030141CSE037 |  |
| Meghana S | 16030141CSE054 |  |
| Naveen Kumar J R | 16030141CSE059 |  |

***ACKNOWLEDGEMENT***

First and foremost, I praise and thank ALMIGTHY GOD whose blessings have bestowed in me the will power and confidence to carry out my project.

I would like to place my heart-felt thanks and gratitude to **Dr. Reeba Korah**, the Dean, ACED, and our Head of the Department **Dr. Shekhar R**, for their constant guidance and encouragement in completing this project work.

I feel it is a pleasure to be indebted to my Project Guide **Prof. Vijaya Lakshmi R** for their invaluable support, advice and encouragement.

I also thank all the staff members of the Department for extending their helping hands to make this Project a success.

I would also like to thank all my friends and my parents who have prayed and helped me during the Project work.

## SYNOPSIS

Data Structure plays an important role when we are processing the data in primary memory. A data structure must be capable of fast retrieval and storage in order to boost the performance of any application. We have come across many such data structures, some of them are Linked List, Stack, Queue, Maps, Trees etc. All the above data structure are only limited up to some extent and cannot go faster than that. The custom data structure will be made based on the existing data structure Binary Search Tree. Compared to other data structure a Binary Search Tree are faster in terms of retrieval and storage. However this data Structure stores based on its property and structure and not based on what type of data is stored in it. Our custom Binary Search Tree will be able to communicate with a machine learning algorithm, which then will calculate priority based on the type of data it Stores and then the custom Binary Search Tree will restructure itself based on this priority. Our custom binary search tree will be generic and store java objects. This objects may contain details inside this details are passed to a machine learning algorithm (MLR – machine Learned Ranking) and then produce a priority value this will be completely automated that is the user need not perform the machine learning manually but our data structure will do it for them. Data structure will be written in java and machine learning will be written in python and these two will communicate using an API between these two languages. The machine learning will be performed at a period of time while operating on the data structure. And more the user uses the data structure the faster the process of retrieval becomes. We did a study on comparison of Map and Binary Search Tree (BST) using complexity formulae for 1000 to 100000 data and saw that the retrieval of data from BST is much faster than a Map and Map is an official data structure of Java. This means that the BST data structure is faster compared to rest. However, BST is only based on its structure and not the data, there is no Machine Learning applied in any data structure although Machine Learning can be used to find the relationship between the data and improve the performance of BST which will result in even better retrieval of data. Our aim is to make a custom Binary Search Tree which will be able to restructure itself in order to make the retrieval faster. This restructuring will be based on machine learned ranking. Also up to my knowledge no one has made an approach to this type of project. In order to demonstrate the data structure we may also have a client side and server side basic code, and the data structure will be running on the server side (if possible).

**CONTENTS**

**ACKNOWLEDGEMENT i**

**SYNOPSIS ii**

**LIST OF FIGURES v**

**LIST OF TABLES vi**

**CHAPTER PAGE NO.**

**CHAPTER 1 1-12**

**INTRODUCTION**

1.1 Statement and Explanation of Basic principle 1

1.2 State of Art 1

1.3 Unsupervised Machine Learning 6

1.4 What has been done in the project? 11

1.5 Chapter-wise Summary 11

**CHAPTER 2 13-14**

**LITERATURE SURVEY**

2.1 Literature Survey 13

**CHAPTER 3 15-16**

**SYSTEM REQUIREMENTS**

3.1 System Requirement 15

3.2 Python 15

3.3 Java 16

**CHAPTER 4 17-19**

**ANALYSIS**

4.1 Existing System 17

4.2 Proposed System 18

**CHAPTER 5 20-35**

**DESIGN AND IMPLEMENATAION**

5.1 Design 21

5.2 Modules Description 22

5.3 Implementing Machine Learning Script 22

5.4 Calculating the Distance for Numeric Data 31

5.5 Calculating the Distance for String Data 33

**CHAPTER 6 36-40**

**TESTING AND RESULTS**

6.1 Testing 36

6.2 Results and Snapshots 38

**CHAPTER 7 41**

**CONCLUSION AND FURTHER WORK**

**BIBILOGRAPHY 42-43**

**LIST OF FIGURES**

**Figure No. Description Page No.**

Fig.1.1 Classification of Data Structure 2

Fig.1.2 Binary Search Tree 4

Fig.1.3 Treap 6

Fig.1.4 k-nearest neighbor 10

Fig.4.1 Simple Array 17

Fig.4.2 Simple Linked List 17

Fig.4.3 Simple Binary Search Tree 18

Fig.4.4 Skewed BST 18

Fig.5.1 Smart Tree Architecture 21

Fig.5.2 Machine Learning Architecture 22

Fig.5.3 Tree Rotation 26

Fig.5.4 Knn Implementation 28

Fig.5.5 Knn Working (a) 28

Fig.5.6 Knn Working (b) 29

Fig.5.7 Spacy Architecture 29

Fig.6.1 Levels of Testing 36

Fig.6.4 System Testing input 1 37

Fig.6.5 System Testing input 2 38

Fig.6.6 Resultant output 38

Fig.6.7 Tree before rotation 39

Fig.6.8 Tree after rotation 40

**LIST OF TABLES**

**Table No. Description Page No.**

Tab.5.8 Data Set 30

Tab.5.9 Numeric Subset 30

Tab.5.10 String Subset 31

Tab.5.11 Numeric Distance 32

Tab.5.12 Numeric Priority 33

Tab.5.13 Text Similarity 34

Tab.6.2 Input csv file for Machine Learning 36

Tab.6.3 Output csv file for Machine Learning 37

**CHAPTER 1**

**INTRODUCTION**

**1.1 STATEMENT AND EXPLANATION OF THE BASIC PRINCIPLE**

Data Structure plays an important role when we are processing the data in primary memory. A data structure must be capable of fast retrieval and storage in order to boost the performance of any application. We have come across many such data structures, some of them are Linked List, Stack, Queue, Maps, Trees etc. All the above data structures are only limited up to some extent and cannot go faster than that. Our aim is to make a custom Binary Search Tree which will be able to restructure itself in order to make the retrieval faster. This restructuring will be based on machine learned ranking.

**1.2 STATE OF ART**

**1.2.1 Data structures**

Data Structure can be defined as the group of data elements which provides an efficient way of storing and organizing data in the computer so that it can be used efficiently. Some examples of Data Structures are arrays, Linked List, Stack, Queue, etc. Data Structures are widely used in almost every aspect of Computer Science i.e. Operating System, Compiler Design, Artificial Intelligence, Graphics and many more.

Data structures are the building blocks of any program or the software. Choosing the appropriate data structure for a program is the most difficult task for a programmer. Following terminology is used as far as data structures are concerned

**Data:** Data can be defined as an elementary value or the collection of values, for example, student's name and its id are the data about the student.

**Group Items:** Data items which have subordinate data items are called Group item, for example, name of a student can have first name and the last name.

**Record:** Record can be defined as the collection of various data items, for example, if we talk about the student entity, then its name, address, course and marks can be grouped together to form the record for the student.

**File:** A File is a collection of various records of one type of entity, for example, if there are 60 employees in the class, then there will be 20 records in the related file where each record contains the data about each employee.

**Attribute and Entity:** An entity represents the class of certain objects. It contains various attributes. Each attribute represents the particular property of that entity.

**Field:** Field is a single elementary unit of information representing the attribute of an entity.

**1.2.2 Importance of data structures**

As applications are getting complex and amount of data is increasing day by day, there may arise the following problems:

**Processor speed:** To handle very large amount of data, high speed processing is required, but as the data is growing day by day to the billions of files per entity, processor may fail to deal with that much amount of data.

**Data search:** Consider an inventory size of 106 items in a store, If our application needs to search for a particular item, it needs to traverse 106 items every time, results in slowing down the search process.

**Multiple requests:** If thousands of users are searching the data simultaneously on a web server, then there are the chances that a very large server can be failed during that process.

In order to solve the above problems, data structures are used. Data is organized to form a data structure in such a way that all items are not required to be searched and required data can be searched instantly.

**1.2.3 Types of data structures**



**Fig.1.1 Classification of Data Structures**

**Linear Data Structure**

A data structure is called linear if all of its elements are arranged in the linear order. In linear data structures, the elements are stored in non-hierarchical way where each element has the successors and predecessors except the first and last element.

Types of Linear Data Structures are given below:

**Arrays**

 An array is a collection of similar type of data items and each data item is called an element of the array. The data type of the element may be any valid data type like char, int, float or double.

The elements of array share the same variable name but each one carries a different index number known as subscript. The array can be one dimensional, two dimensional or multidimensional.

The individual elements of the array age are:

age[0], age[1], age[2], age[3],......... age[98], age[99].

**Linked list**

Linked list is a linear data structure which is used to maintain a list in the memory. It can be seen as the collection of nodes stored at non-contiguous memory locations. Each node of the list contains a pointer to its adjacent node.

**Stack**

Stack is a linear list in which insertion and deletions are allowed only at one end, called **top**. A stack is an abstract data type (ADT), can be implemented in most of the programming languages. It is named as stack because it behaves like a real-world stack, for example: - piles of plates or deck of cards etc.

**Queue**

 Queue is a linear list in which elements can be inserted only at one end called **rear** and deleted only at the other end called **front**. It is an abstract data structure, similar to stack. Queue is opened at both end therefore it follows First-In-First-Out (FIFO) methodology for storing the data items.

**Non linear data structures**

 This data structure does not form a sequence i.e. each item or element is connected with two or more other items in a non-linear arrangement. The data elements are not arranged in sequential structure. Types of Non Linear Data Structures are given below:

**Trees**

Trees are multilevel data structures with a hierarchical relationship among its elements known as nodes. The bottommost nodes in the hierarchy are called **leaf node** while the topmost node is called **root node**. Each node contains pointers to point adjacent nodes.

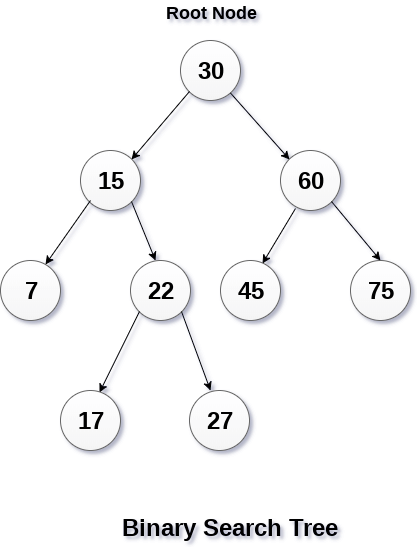
Tree data structure is based on the parent-child relationship among the nodes. Each node in the tree can have more than one children except the leaf nodes whereas each node can have at most one parent except the root node. Trees can be classified into many categories which will be discussed later in this tutorial.

**Graphs**

Graphs can be defined as the pictorial representation of the set of elements (represented by vertices) connected by the links known as edges. A graph is different from tree in the sense that a graph can have cycle while the tree cannot have the one.

**1.2.4 Binary search tree**

Binary Search tree can be defined as a class of binary trees, in which the nodes are arranged in a specific order. This is also called ordered binary tree. In a binary search tree, the value of all the nodes in the left sub-tree is less than the value of the root. Similarly, value of all the nodes in the right sub-tree is greater than or equal to the value of the root. This rule will be recursively applied to all the left and right sub-trees of the root.



**Fig.1.2 Binary Search Tree**

**Advantages of Binary Search Tree**

* Searching become very efficient in a binary search tree since, we get a hint at each step, about which sub-tree contains the desired element.
* The binary search tree is considered as efficient data structure in compare to arrays and linked lists. In searching process, it removes half sub-tree at every step. Searching for an element in a binary search tree takes o(log2n) time. In worst case, the time it takes to search an element is 0(n).
* It also speed up the insertion and deletion operations as compare to that in array and linked list.

**1.2.5 Treap**

The **Treap** and the **Randomized Binary Search Tree** are two closely related forms of binary search tree data structures that maintain a dynamic set of ordered keys and allow binary searches among the keys. After any sequence of insertions and deletions of keys, the shape of the tree is a random variable with the same probability distribution as a random binary tree in particular, with high probability its height is proportional to the logarithm of the number of keys, so that each search, insertion, or deletion operation takes logarithmic time to perform.

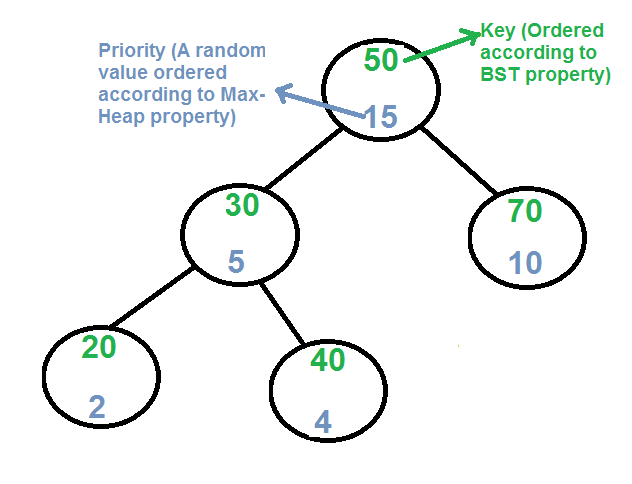
Treaps support the following basic operations:

* To search for a given key value, apply a standard [binary search algorithm](https://en.wikipedia.org/wiki/Binary_search_algorithm) in a binary search tree, ignoring the priorities.
* To insert a new key *x* into the treap, generate a random priority *y* for *x*. Binary search for *x* in the tree, and create a new node at the leaf position where the binary search determines a node for *x* should exist. Then, as long as *x* is not the root of the tree and has a larger priority number than its parent *z*, perform a [tree rotation](https://en.wikipedia.org/wiki/Tree_rotation) that reverses the parent-child relation between *x* and *z*.
* To delete a node *x* from the treap, if *x* is a leaf of the tree, simply remove it. If *x* has a single child *z*, remove *x* from the tree and make *z* be the child of the parent of *x* (or make *z* the root of the tree if *x* had no parent). Finally, if *x* has two children, swap its position in the tree with the position of its immediate successor *z* in the sorted order, resulting in one of the previous cases. In this final case, the swap may violate the heap-ordering property for *z*, so additional rotations may need to be performed to restore this property.

**Bulk operations**

In addition to the single-element insert, delete and lookup operations, several fast "bulk" operations have been defined on treaps namely [union](https://en.wikipedia.org/wiki/Union_(set_theory)), [intersection](https://en.wikipedia.org/wiki/Intersection_(set_theory)) and [set difference](https://en.wikipedia.org/wiki/Set_difference). These rely on two helper operations, split and join.

* To split a treap into two smaller treaps, those smaller than key x, and those larger than key x, insert x into the treap with maximum priority—larger than the priority of any node in the treap. After this insertion, x will be the root node of the treap, all values less than x will be found in the left subtreap, and all values greater than x will be found in the right subtreap. This costs as much as a single insertion into the treap.
* Joining two treaps that are the product of a former split, one can safely assume that the greatest value in the first treap is less than the smallest value in the second treap. Create a new node with value x, such that x is larger than this max-value in the first treap, and smaller than the min-value in the second treap, assign it the minimum priority, then set its left child to the first heap and its right child to the second heap. Rotate as necessary to fix the heap order. After that it will be a leaf node, and can easily be deleted. The result is one treap merged from the two original treaps. This is effectively "undoing" a split, and costs the same. More generally, the join operation can work on two treaps and a key with arbitrary priority (i.e., not necessary to be the highest).



**Fig.1.3 Treap**

**1.3 UNSUPERVISED RANK AGGRIGATION**

Many applications in information retrieval, natural language processing, data mining, and related fields require a ranking of instances with respect to a specified criteria as opposed to a classification. Furthermore, for many such problems, multiple established ranking models have been well studied and it is desirable to combine their results into a joint ranking, a formalism denoted as rank aggregation. Unsupervised Rank Aggregation returns a linear combination of the individual ranking functions based on the principle of rewarding ordering agreement between the rankers. Ranking items is a fundamental computer and information sciences problem. Most closely associated with information retrieval, ranking has recently attracted significant attention from the machine learning and natural language processing communities. While classification is the standard task of inductive learning, many applications require the expressivity of ranking. A related, but less thoroughly studied problem is rank aggregation, where multiple existing rankings of an item set are combined into a joint ranking. In the information retrieval community, this is the data fusion problem and corresponds to deriving a document ranking based on the input of multiple retrieval systems. For domains where ranking algorithms exist which utilize different modalities or views over the data, rank aggregation is particularly appealing as these views are difficult to combine into a single model. From a machine learning perspective, this work is most ostensibly related to which extends ideas regarding expert ensembles and boosting to ranking. In addition to a different model representation, the fundamental difference is that our algorithm is an unsupervised learning algorithm. Another related vein is the study of deriving voting policies which satisfy specified axiomatic properties(e.g. the independence of irrelevant alternatives) . Unsupervised Ranking algorithm is similar in that the input is a set of ranking functions and no supervised training is required. However, it adaptively learns a parameterized linear combination to optimize the relative influence of individual rankers.

**1.3.1 Text similarity**

Similarity is determined by comparing **word vectors** or “word embeddings”, multi-dimensional meaning representations of a word. Word vectors can be generated using an algorithm like word2vec and usually look like this:

BANANA.VECTOR

array([2.02280000e-01, -7.66180009e-02, 3.70319992e-01,

3.28450017e-02, -4.19569999e-01, 7.20689967e-02,

-3.74760002e-01, 5.74599989e-02, -1.24009997e-02,

5.29489994e-01, -5.23800015e-01, -1.97710007e-01,

-3.41470003e-01, 5.33169985e-01, -2.53309999e-02,

1.73800007e-01, 1.67720005e-01, 8.39839995e-01,

5.51070012e-02, 1.05470002e-01, 3.78719985e-01,

2.42750004e-01, 1.47449998e-02, 5.59509993e-01,

1.25210002e-01, -6.75960004e-01, 3.58420014e-01,

# ... and so on ...

3.66849989e-01, 2.52470002e-03, -6.40089989e-01,

-2.97650009e-01, 7.89430022e-01, 3.31680000e-01,

-1.19659996e+00, -4.71559986e-02, 5.31750023e-01], dtype=float32)

**Note**:

To make them compact and fast, spaCy’s small models (all packages that end in sm) don’t ship with word vectors, and only include context-sensitive tensors. This means you can still use the similarity() methods to compare documents, spans and tokens – but the result won’t be as good, and individual tokens won’t have any vectors assigned. So in order to use real word vectors, you need to download a larger model:

- python -m spacy download en\_core\_web\_sm

+ python -m spacy download en\_core\_web\_lg

Models that come with built-in word vectors make them available as the Token.vector attribute. Doc.vector and Span.vector will default to an average of their token vectors. You can also check if a token has a vector assigned, and get the L2 norm, which can be used to normalize vectors.

import spacy

nlp = spacy.load("en\_core\_web\_md")

tokens = nlp("dog cat banana afskfsd")

for token in tokens:

print(token.text, token.has\_vector, token.vector\_norm, token.is\_oov)

**Output:**

dog True 7.0336733 False

cat True 6.6808186 False

banana True 6.700014 False

afskfsd False 0.0 True

The words “dog”, “cat” and “banana” are all pretty common in English, so they’re part of the model’s vocabulary, and come with a vector. The word “afskfsd” on the other hand is a lot less common and out-of-vocabulary – so its vector representation consists of 300 dimensions of 0, which means it’s practically nonexistent. If your application will benefit from a large vocabulary with more vectors, you should consider using one of the larger models or loading in a full vector package, for example, en\_vectors\_web\_lg, which includes over 1 million unique vectors.

spaCy is able to compare two objects, and make a prediction of how similar they are. Predicting similarity is useful for building recommendation systems or flagging duplicates. For example, you can suggest a user content that’s similar to what they’re currently looking at, or label a support ticket as a duplicate if it’s very similar to an already existing one.

Each Doc, Span and Token comes with a .similarity() method that lets you compare it with another object, and determine the similarity. Of course similarity is always subjective – whether “dog” and “cat” are similar really depends on how you’re looking at it. spaCy’s similarity model usually assumes a pretty general-purpose definition of similarity.

import spacy

nlp = spacy.load("en\_core\_web\_md") # make sure to use larger model!

tokens = nlp("dog cat banana")

for token1 in tokens:

for token2 in tokens:

print(token1.text, token2.text, token1.similarity(token2))

**OUTPUT:**

dog dog 1.0

dog cat 0.80168545

dog banana 0.24327643

cat dog 0.80168545

cat cat 1.0

cat banana 0.28154364

banana dog 0.24327643

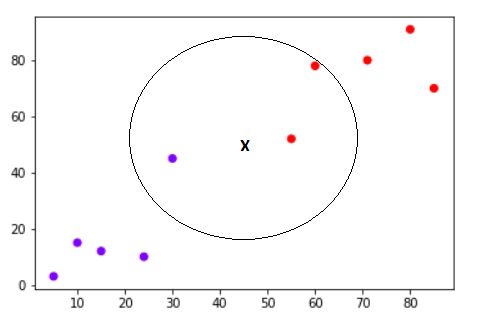
banana cat 0.28154364

banana banana 1.0

In this case, the model’s predictions are pretty on point. A dog is very similar to a cat, whereas a banana is not very similar to either of them. Identical tokens are obviously 100% similar to each other (just not always exactly 1.0, because of vector math and floating point imprecisions).

**1.3.2 Distance based priority**

Distance Based Priority is basically the idea of finding the distance between the values. For example, if we consider the age groups a1=6, a2=18 and a3=30. If we calculate the distance between the age groups then, the distance between the age groups a1 and a2 is 1 and the distance between the age group a2 and a3 is 12. So, the age groups a1 and a2 are closely related to each other compared to the age groups a1 and a3 or a2 and a3. If the distance is less then, the priority will be more and if the priority is more it means that they are closely related to each other. If the distance is more then, the priority is less which means that they are not closely related to each other. To calculate the accurate value inversion is also takes place. Python provides a library called **nearest neighbor** which can be used to calculate the nearest neighbor distance between the values.



**Fig.1.4 K nearest neighbor**

**1.4 WHAT HAS BEEN DONE IN THE PROJECT?**

Our custom Binary Search Tree will be able to communicate with a machine learning algorithm, which then will calculate priority based on the type of data it Stores and then the custom Binary Search Tree will restructure itself based on this priority. Our custom binary search tree will be generic and store java objects. This objects may contain details inside this details are passed to a machine learning algorithm (MLR – machine Learned Ranking) and then produce a priority value this will be completely automated that is the user need not perform the machine learning manually but our data structure will do it for them. Data structure will be written in java and machine learning will be written in python and these two will communicate using an API between these two languages. The machine learning will be performed at a period of time while operating on the data structure. And more the user uses the data structure the faster the process of retrieval becomes. The idea here is to combine 2 separate technologies to perform operations faster.

**1.5 CHAPTER WISE SUMMARY OF THE PROJECT**

The **First Chapter** shows the statement and the basic principle of the theme of the subject of the project work. The “State of Art” gives the basic idea about the data structures used and the “Unsupervised Rank Aggregation” gives the basic idea of how the ranking system is used. Then it explains about what work is done in the project.

In the **Second Chapter** we review the various research papers that have contributed various methodologies to calculate and process the necessary modules of our project. It has experimental ideas and tested ones that have been a great help to develop the application.

In the **Third Chapter** we look at the general system requirements that are necessary run our application. This also allows an insight into the technologies used in the development of the application.

In the **Forth Chapter** we took a brief look at all the frequently used data structures and also we had a brief understanding of their advantages and disadvantages. After getting an idea how data is stored and manipulated in various different data structures, we can finally see that the binary search was the most optimum way to store and retrieve data for use, but we also encountered the anomaly where the tree might end up being skewed as the tree usually stores the data based on a simple comparison between the elements. To overcome this and further improve the access time of the requested data we make use if a smart data structure which dynamically performs tree rotations to pervert the skewing of the trees and optimizes the access times.

The **Fifth Chapter** speaks about how java system and the machine learning script works together to make a dynamic data structure, namely smart structure that will fetch the requested data in form of a modified binary search tree that uses machine learning to calculate the priority in which the java objects should be accessed. This can be achieved by integrating a python API within the java system and pass the data to the python script using a CSV file, which calculates the priority of the data objects for both string and numeric data types and finally using the same API we read the priority values to restructure the dynamic binary search tree by performing rotations to enable maximum efficiency.

The **Sixth Chapter** deals with how after the testing and validation process we can see that after mimicking the behavior of a server environment, by entering the details of 5 students into our data structure and then searched every object for a number of times in order to increase the visits of the every node. After this we can look at the triggered system logs to check that all the components of the project are working as required.

For the validation process of the code we run the entire code base in debug mode in which we can observe that the initially null priority values are later populated after running the machine learning script, and we can conclude the validation after verifying the tree rotations and checking for any anomalous skewing in the tree. It also contains the snapshots of the results obtained during the testing and validation process.

The **Seventh Chapter** speaks about the conclusion of the project and further scope that can be carried out from our project.

**CHAPTER 2**

**LITERATURE SURVEY**

**2.1 LITERATURE SURVEY**

**2.1.1 Title: Unsupervised Rank Aggregation with Distance-Based Models**

**Author: Alexandre Klementiev, Dan Roth, Kevin Small**

**Abstract:** The need to meaningfully combine sets of rankings often comes up when one deals with ranked data. Although several heuristic and supervised learning approaches to rank aggregation exist, they require domain knowledge or supervised ranked data, both of which are expensive to acquire. In order to address these limitations, we propose a mathematical and algorithmic framework for learning to aggregate (partial) rankings without supervision. We instantiate the frame work for the cases of combining permutations and combining top-k lists, and propose a novel metric for the latter. Experiments in both scenarios demonstrate the effectiveness of the proposed formalism.

**2.1.2 Title: Combining feature ranking algorithms through rank aggregation**

**Author: Ronaldo C. Prati**

**Abstract:** The problem of combining multiple feature rankings into a more robust ranking is investigated. A general framework for ensemble feature ranking is proposed, alongside four instantiations of this framework using different ranking aggregation methods. An empirical evaluation using 39 UCI datasets, three different learning algorithms and three different performance measures enable us to reach a compelling conclusion: ensemble feature ranking do improve the quality of feature rankings. Furthermore, one of the proposed methods was able to achieve results statistically significantly better than the others.

**2.1.3 Title: An Unsupervised Learning algorithm for rank aggregation**

**Author: Alexandre Klementiev, Dan Roth, Kevin Small**

**Abstract:** Many applications in information retrieval, natural language processing, data mining, and related fields require a ranking of instances with respect to a specified criteria as opposed to a classification. Furthermore, for many such problems, multiple established ranking models have been well studied and it is desirable to combine their results into a joint ranking, a formalism denoted as rank aggregation. This work presents a novel unsupervised learning algorithm for rank aggregation (ULARA) which returns a linear combination of the individual ranking functions based on the principle of rewarding ordering agreement between the rankers. In addition to presenting ULARA, it demonstrates its effectiveness on a data fusion task across ad hoc retrieval systems.

**2.1.4 Title: A Statistical Convergence Perspective of algorithms for rank aggregation from pair wise data**

**Author: Arun Rajkumar, Shivani Agarwal**

**Abstract:** There has been much interest recently in the problem of rank aggregation from pair wise data. A natural question that arises is: under what sorts of statistical assumptions do various rank aggregation algorithms converge to an ‘optimal’ ranking? In this paper, we consider this question in a natural setting where pair wise comparisons are drawn randomly and independently from some underlying probability distribution. We first show that, under a ‘time-reversibility’ or Bradley-Terry-Luce (BTL) condition on the distribution, the rank centrality (PageRank) and least squares (HodgeRank) algorithms both converge to an optimal ranking. Next, we show that a matrix version of the Borda count algorithm, and more surprisingly, an algorithm which performs maximum likelihood estimation under a BTL assumption, both converge to an optimal ranking under a ‘low-noise’ condition that is strictly more general than BTL. Finally, we propose a new SVM-based algorithm for rank aggregation from pair wise data, and show that this converges to an optimal ranking under an even more general condition that we term ‘generalized low-noise’. In all cases, we provide explicit sample complexity bounds for exact recovery of an optimal ranking. Our experiments confirm our theoretical findings and help to shed light on the statistical behavior of various rank aggregation algorithms.

**2.1.5 Title: Spectral Feature Selection for supervised and unsupervised learning**

**Author: Zheng Zhao, Huan Liu**

**Abstract**: Feature selection aims to reduce dimensionality for building comprehensible learning models with good generalization performance. Feature selection algorithms are largely studied separately according to the type of learning: supervised or unsupervised. This work exploits intrinsic properties underlying supervised and unsupervised feature selection algorithms, and proposes a unified framework for feature selection based on spectral graph theory. The proposed framework is able to generate families of algorithms for both supervised and unsupervised feature selection. And we show that existing powerful algorithms such as Relief (supervised) and Laplacian Score (unsupervised) are special cases of the proposed framework. To the best of our knowledge, this work is the first attempt to unify supervised and unsupervised feature selection, and enable their joint study under a general framework. Experiments demonstrated the efficacy of the novel algorithms derived from the framework.

**CHAPTER 3**

**SYSTEM REQUIREMENTS**

**3.1 SYSTEM REQUIREMENT**

**3.1.1 Software Requirements:**

* Operating system : Windows 10, Ubuntu
* Coding Language : Python 3, Java 8

**3.1.2 Hardware Requirements:**

* System : Intel Core i3 and above
* Hard Disk : 512GB
* Monitor : 15’’LED
* Input Device : Keyboard , Mouse
* Ram : 8GB DDR4

**3.2 PYTHON**

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aims to help programmers write clear, logical code for small and large-scale projects. Van Rossum shouldered sole responsibility for the project until July 2018 but now shares his leadership as a member of a five-person steering council.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source reference implementation. A non-profit organization, the Python Software Foundation, manages Python and CPython.

**3.2.1 Python Standard Library**

While the Python Language Reference describes the exact syntax and semantics of the Python language, this library reference manual describes the standard library that is distributed with Python. It also describes some of the optional components that are commonly included in Python distributions.

Python’s standard library is very extensive, offering a wide range of facilities as indicated by the long table of contents listed below. The library contains built-in modules (written in C) that provide access to system functionality such as file I/O that would otherwise be inaccessible to Python programmers, as well as modules written in Python that provide standardized solutions for many problems that occur in everyday programming. Some of these modules are explicitly designed to encourage and enhance the portability of Python programs by abstracting away platform-specifics into platform-neutral APIs.

The Python installers for the Windows platform usually include the entire standard library and often also include many additional components. For Unix-like operating systems Python is normally provided as a collection of packages, so it may be necessary to use the packaging tools provided with the operating system to obtain some or all of the optional components.

**3.3JAVA**

**What is Java?**

Java is a popular programming language, created in 1995.It is owned by Oracle, and more than 3 billion devices run Java.

It is used for:

* Mobile applications (especially Android apps)
* Desktop applications
* Web applications
* Web servers and application servers
* Games
* Database connection
* And much, much more!

**Why Use Java?**

* Java works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)
* It is one of the most popular programming language in the world
* It is easy to learn and simple to use
* It is open-source and free
* It is secure, fast and powerful
* It has a huge community support (tens of millions of developers)
* Java is an object-oriented language which gives a clear structure to programs and allows code to be reused, lowering development costs
* As Java is close to C++ and C#, it makes it easy for programmers to switch to Java or vice versa.

**CHAPTER 4**

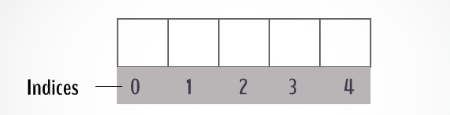
**ANALYSIS**

**4.1 EXISTING SYSTEM**

As we all know data structure plays an important role in real time processing of data but when it comes to large amounts of data our collection library does not provide us a faster or efficient data structure for faster insertion and retrieval of data.

So most of the huge IT companies tend to develop their own data structure like binary search tree or another custom tree based on their needs. The problem arises because all the data structure stores the data based on what the data structure is defined as and not what type of data is coming. For instance let us take some of the data structure offered by collection library and see where the problem arises.

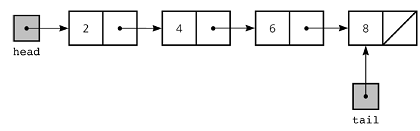
**4.1.1 Arrays**



**Fig.4.1 Simple Array**

Arrays are by far the most commonly used data structure and can store retrieve data faster than any other data structure , but the problem comes when we have a huge amount of data, since array needs continuous block of memory in order to store the data and in order to use an array we have to know the amount of data that will be stored in array in prior so when it comes to a large amount of data say a million data , array fails there because the memory may not have that big continuous data since there will be fragments in memory which are not cleaned.

**4.1.2 Linked lists**

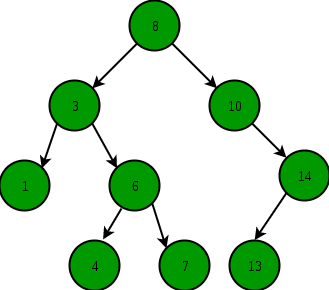


**Fig.4.2 Simple Linked List**

Linked list helps in solving the problem that is faced by arrays, we can store large amount of data with ease without need of finding continuous memory as one data will have the address to another. But the problem with this data structure is that insertion and retrieval will take a time complexity of O(n) where n stands for number of elements that means in order to search for millionth element we will need an order of million time.

**4.1.3 Binary search tree**

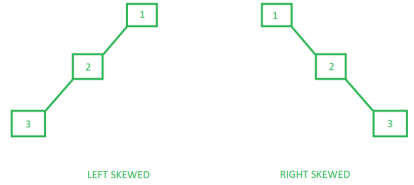
Binary search tree is way better than any other data structure available out there and gives acceptable time complexity compared to other data structure, the time complexity of BST is O(h) where h is in range log(n) to n where n is the number of data. However there are some downsides of BST which are shown below.



**Fig.4.3 Simple Binary Search Tree**

A typical BST looks something like this, there is a root node and have child elements in left and right node. And the BST follows the same structure for any type of data coming into the tree irrespective of what type of data is coming in.

The problem here is that the BST stores the data based on its own structure and not depending on what type of data is coming in or how it is related to each other just that the value must be comparatively be greater or smaller to each other.



**Fig.4.4 Skewed BST**

As seen above this is an example of a skewed BST , now suppose there is a test case where the user always needs to access the 3rd element , this will take a time complexity of O(3) every time for the user , which is not so healthy.

**4.2 PROPOSED SYSTEM**

In our project we have a custom-made binary search tree which will have machine learning integrated into it which will automatically be updating the BST structure based on how the data is visited and how the data is related to each other.

**4.2.1 Explanation**

Let us take the simple example of above scenario in a skewed BST when the 3rd element is accessed multiple times then our BST will understand that and rotate the tree such that the 3rd element comes to 1st and all the elements related to it will be around it, also the BST structure will also remain consistent and follow the standard BST property.

This will in turn ensure that next time this data will be accessed it will only take O(1) time and hence it will be faster than the standard binary search tree.

**4.2.2 Performance analysis**

The performance of our system is better than that of standard binary search tree and the time complexity of our system will be less than or equal to O(h) where h is in range log(n) and n where n represents the number of elements , however over time of using this system the structure of the BST will become better and better which will result in concentrating the log(n) side of the complexity.

**CHAPTER 5**

**DESIGN AND IMPLEMENTATION**

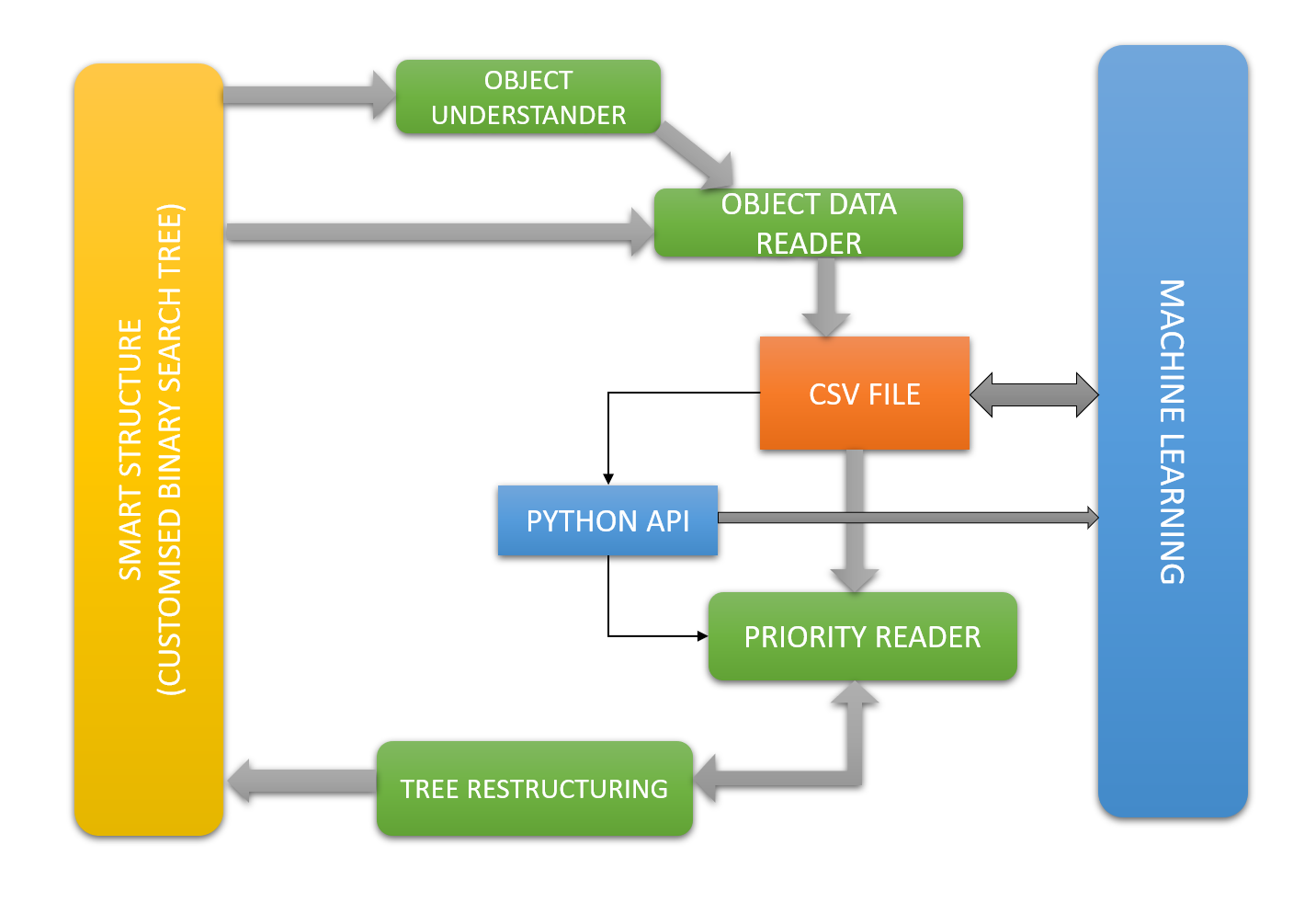
Data Structure plays an important role when we are processing the data in primary memory. A data structure must be capable of fast retrieval and storage in order to boost the performance of any application. We have come across many such data structures, some of them are Linked List, Stack, Queue, Maps, Trees etc. All the above data structure are only limited up to some extent and cannot go faster than that. Our aim is to make a custom Binary Search Tree which will be able to restructure itself in order to make the retrieval faster. This restructuring will be based on machine learned ranking. The custom data structure will be made based on the existing data structure Binary Search Tree. Compared to other data structure a Binary Search Tree are faster in terms of retrieval and storage. However this data Structure stores based on its property and structure and not based on what type of data is stored in it. Our custom Binary Search Tree will be able to communicate with a machine learning algorithm, which then will calculate priority based on the type of data it Stores and then the custom Binary Search Tree will restructure itself based on this priority.Our custom binary search tree will be generic and store java objects. This objects may contain details inside this details are passed to a machine learning algorithm (MLR – machine Learned Ranking) and then produce a priority value this will be completely automated that is the user need not perform the machine learning manually but our data structure will do it for them. Data structure will be written in java and machine learning will be written in python and these two will communicate using an API between these two languages. The machine learning will be performed at a period of time while operating on the data structure. And more the user uses the data structure the faster the process of retrieval becomes. The idea here is to combine 2 separate technologies to perform operations faster.

Until now the existing data structures just stores the data based on their structure , which makes them limited but our data structure will be learning as the user keeps using the data structure more and more, which in turn makes the processing faster and faster. A faster data structure will ensure that the application that the user is making will be more optimized. Currently people don’t even use standard Binary Search Tree which has way more efficiency compared to other data structure, they use the standard collection library provided by java that is maps, linked list etc.

**5.1 DESIGN**

**5.1.1 System Architecture**

**A. Smart Tree Architecture**

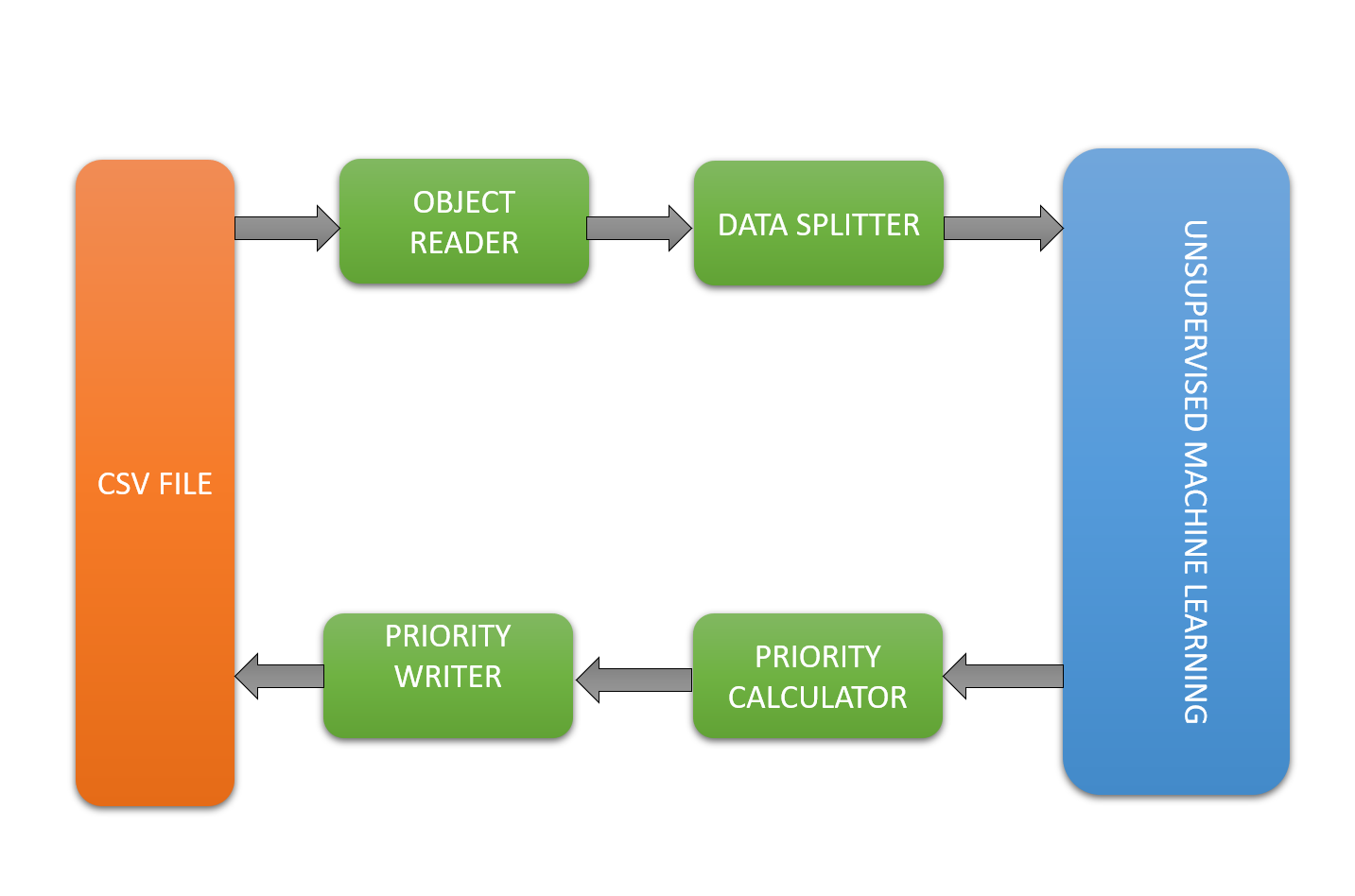
****

**Fig.5.1 Smart Tree Architecture**

The above architecture illustrates the design of the smart tree which has 7 modules:

1. Custom binary search tree
2. Object understander
3. Object data writer
4. Python api
5. Priority reader
6. Priority setter
7. Tree restructuring

**B. Machine Learning**

****

**Fig.5.2 Machine Learning Architecture**

The above architecture illustrates the design of the priority calculating script which has 5 modules:

1. Object data reader
2. Data partitioning
3. Distance based unsupervised learning
4. Priority calculator
5. Priority writer

**5.2 MODULES DESCRIPTION:**

**5.2.1 Smart tree**

Smart tree will be a class whose object can be created and used in order to insert and retrieve data from. A basic collections library data structure usage also has a same method.

All the components will rest in jar file which will contain all the modules necessary for proper functioning of our data structure. Whereas in order to use our data structure the user will have to load the library into their project and then they can make use of it along with this the smartTree class will have some more methods which will help in proper functioning of our system the list of some of the methods.

The node class has some more properties compared to a standard binary search tree, it has visits and priority, the visits represents the number of times that particular node has been called and priority is set when the machine learning completes its task.

**5.2.2 Object Understander**

Basically, our data structure is generic as the rest of them and this means that the user will have the authority to store any sort of data possible into the data structure. This means that in order to perform machine learning on the data we will have to know what type of data comes in, this will be ensured by object understander.

Once the object understander will read what type data the user wants to store, then it will forward the information to Object Data Reader module then the object data reader module will do its work based on the given information.

In order to achieve this, we are going to use reflection library provided by java library, this library will allow us to perform necessary operations on java objects. For our project we need to get the fields of object in order to perform machine learning on it, for this we call the getClassFields() method of the Class. This will give us an array of field objects which belongs to reflection library. This field object array will be passed on to object data reader module in order to use it to read the properties of the object. The manager class is the one where all the machine learning process is managed. It will have all the modules of java part. We see that the ObjectToCSV object is object data reader here and at line 23 the setClassFields will set the fields object array as a property on objetToCSVString object; this will later on be used to read the data of the objects.

**5.2.3 Object data reader**

The object data reader will read all the data present in the binary search tree and store this data in the form of comma separated values. Once this will be done this csv values will be written to a csv file. This is the method to store the data to list of comma separated values for a particular node. This method is present in the objectToCSV class. This method will store the data for one node, and this will iterate over the complete binary search tree using the following method. This is a general syntax of lambda expression in java 8 and this is just passing the storeFieldData method to the performOperations method in order to execute it, the perform operations method definition is given below.

The performOperation is basically taking the method and executing on the current node, after that it will recurse the function until all the nodes are used for execution.

Once the data is read and stored into the list it is time to store this data into the csv file this csv file is created by the csvFileIO class and is present in the users current directory this path will later on provided to the python API to execute the machine learning script on the csv file.

**5.2.4 Python API**

The python API will ensure that the once the data has been written to csv file the machine learning script is called along with the path to csv file for the machine learning script reference. It will also ensure that the newly obtained priority values will be used to rotate the binary search tree to increase the access times of the requested data.

The Manager interface is used control and direct the overall flow of the python API. It defines 3 main functions

1. public void initializeLearning(SmartBST<T> tree,Class<?> class1)throws Exception;
2. public void learn(SmartBST<T> tree) throws Exception;
3. public void endLearning();

The DefaultManager class implements the Manager interface to execute all the necessary instructions from passing the data to the python script which houses the machine learning algorithm to getting the priority from the script and converting it into the object format to perform the required tree rotations.

**5.2.5 ML executor**

We make use of the DefaultMLExecutor to load and execute both the Machine learning Python script and the required CSV file along with it.

The DefaultMLExecutor has 2 functions that carry out the required operations

1. private void validate(String mlFilePath,StringcsvFilePath) throws SmartBSTException.

The validate function takes the mlFilePath and the csvFilePath as its input parameters in form of a string and makes sure that they’re not empty.

1. public void execute(String mlFilePath,StringcsvFilePath) throws Exception

The execute function takes the mlFilePath and the csvFilePath as its input parameters in form of a string and uses the ProcessBuilder class to start the script.

But to execute the python script needs to have the required data in correct format as the python script can’t read the java objects, we make use of the following methods to load the data into a CSV file and pass it along to the script so that the python script can calculate the priority and return it in the same format.

**5.2.6 CSVFileIO**

We use the CSVFileIO interface to define 5 functions:

1. public unctiocreateCSVFile(String unction) throws Exception;

It uses the System.getProperty() to obtain the necessary user directory and makes use of the File object to create the CSV file.

1. public List<String>readFromCSVFile();

We make use of the CustomFileReader and its read() function to read the CSV file.

It returns the data in form of a list.

1. public void writeToCSVFile(List<String> data);

We make use of the custom file writer to write the data onto the CSV file.

1. public unctiodeleteCSVFile();

We use the delete() function to delete the required CSV file.

1. public String getCSVFilePath() throws Exception ;

We use the getCanonicalPath() function to obtain the CSV file path.

**5.2.7 CSVToObject**

The CSVToObject section is used to obtain the data from the CSV file using the previously defined IO functions and use the priorities outputted by the machine learning file and store it in the object for further use of restructuring the tree.

We use the CSVToObject interface to define 2 functions:

1. public void setCSVData(List<String> data)

It makes use of a LineConverter that has the tokenise() function that creates a stream of tokens that can be used to obtain the priorities in from of map objects.

1. public void setPriority(Node<T> node)

This function finally writes the values of the priorities to the nodes of the binary search tree.

**5.2.8 API manager**

The DefaultManager class implements the Manager interface to execute all the necessary instructions from passing the data to the python script which houses the machine learning algorithm to getting the priority from the script and converting it into the object format to perform the required tree rotations.

1. public void learn(SmartBST<T> tree) throws Exception

The initiates the previously defined methods in which it reads the object data, copies it to the CSV file and passes it along with the script to execute the script. After execution it also initiates the restructuring of the tree after obtaining the priority values.

1. public void endLearning()

This method is to stop the learning process and delete the temporary CSV file.

1. public void run()

This method calls the learn() method and handles any errors or exceptions.

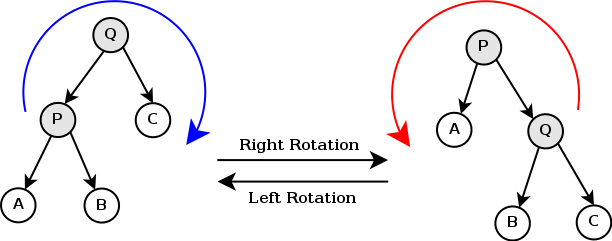
1. protected void finalize() throws Throwable

This method simply finalizes the learning process and calls the endLearning() method.

**5.2.9 Tree Rotations**

In discrete mathematics, tree rotation is an operation on a binary tree that changes the structure without interfering with the order of the elements. A tree rotation moves one node up in the tree and one node down. It is used to change the shape of the tree, and in particular to decrease its height by moving smaller subtrees down and larger subtrees up, resulting in improved performance of many tree operations.

There exists an inconsistency in different descriptions as to the definition of the direction of rotations. Some say that the direction of rotation reflects the direction that a node is moving upon rotation (a left child rotating into its parent’s location is a right rotation) while others say that the direction of rotation reflects which subtree is rotating (a left subtree rotating into its parent’s location is a left rotation, the opposite of the former). This article takes the approach of the directional movement of the rotating node.



**Fig.5.3 Tree Rotations**

The right rotation operation as shown in the adjacent image is performed with Q as the root and hence is a right rotation on, or rooted at, Q. This operation results in a rotation of the tree in the clockwise direction. The inverse operation is the left rotation, which results in a movement in a counter-clockwise direction (the left rotation shown above is rooted at P). The key to understanding how a rotation functions is to understand its constraints. In particular the order of the leaves of the tree (when read left to right for example) cannot change (another way to think of it is that the order that the leaves would be visited in an in-order traversal must be the same after the operation as before). Another constraint is the main property of a binary search tree, namely that the right child is greater than the parent and the left child is less than the parent. Notice that the right child of a left child of the root of a sub-tree (for example node B in the diagram for the tree rooted at Q) can become the left child of the root, that itself becomes the right child of the “new” root in the rotated sub-tree, without violating either of those constraints. As you can see in the diagram, the order of the leaves doesn’t change. The opposite operation also preserves the order and is the second kind of rotation.

Assuming this is a binary search tree, as stated above, the elements must be interpreted as variables that can be compared to each other. The alphabetic characters to the left are used as placeholders for these variables. In the animation to the right, capital alphabetic characters are used as variable placeholders while lowercase Greek letters are placeholders for an entire set of variables. The circles represent individual nodes and the triangles represent subtrees. Each subtree could be empty, consist of a single node, or consist of any number of nodes.

**5.1.10 Restructure**

We now make use of the restructure() method to compare the leaf nodes and the rotate the tree as dictated by the priority that we obtained from the machine learning script.We use the getLeafNodes() method to obtain the leaf nodes of the current node to execute the tree restructuring process.

**5.2.11 Rotate**

The rotate() method does a simple comparison of the left and the right nodes and decides whether the rotation is necessary. As discussed earlier by making a left or a right rotation as deemed necessary we prevent the occurrence of a skewed tree.

**5.3 IMPLEMENTING MACHINE LEARNING SCRIPT**

**5.3.1 Importing the necessary libraries**

**import pandas as pd;**

Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language

**import numpy as np;**

NumPy is a python library used for working with arrays. It also has functions for working in domain of linear algebra, Fourier transform, and matrices.

**From pandas import Series**

Pandas Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called index. Pandas Series is nothing but a column in an excel sheet. Labels need not be unique but must be a hashable type.

**From pandas import DataFrame**

Two-dimensional, size-mutable, potentially heterogeneous tabular data. Data structure also contains labeled axes (rows and columns). Arithmetic operations align on both row and column labels. Can be thought of as a dict-like container for Series objects. The primary pandas data structure.

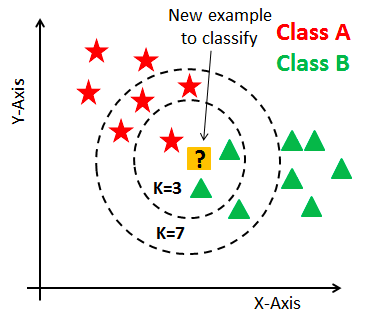
**From sklearn.neighbors import NearestNeighbors**

k-nearest neighbors (knn) is a non-parametric method used for classification and regression. In both cases, 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. KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. To evaluate any technique we generally look at 3 important aspects:

1. Ease to interpret output

2. Calculation time

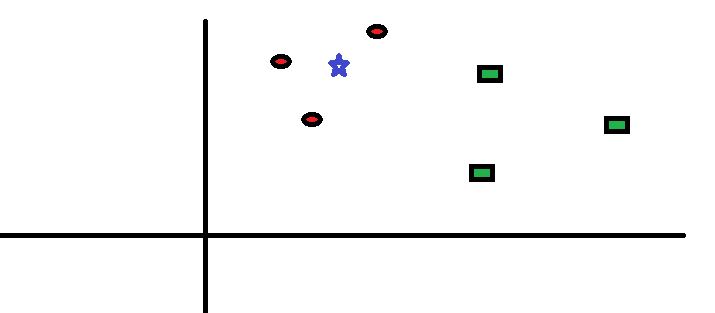
3. Predictive Power



**Fig.5.4 KNN Implementation**

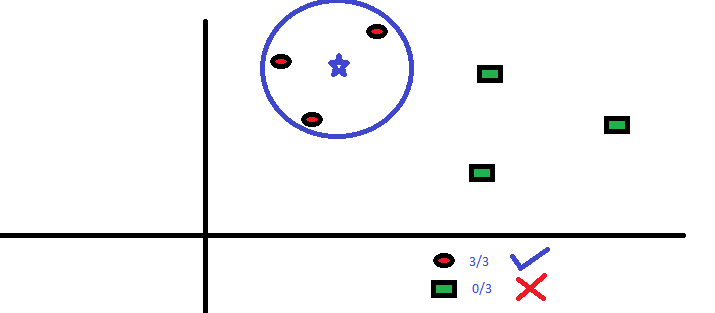
**How does the KNN algorithm work?**

Let’s take a simple case to understand this algorithm. Following is a spread of red circles (RC) and green squares (GS):



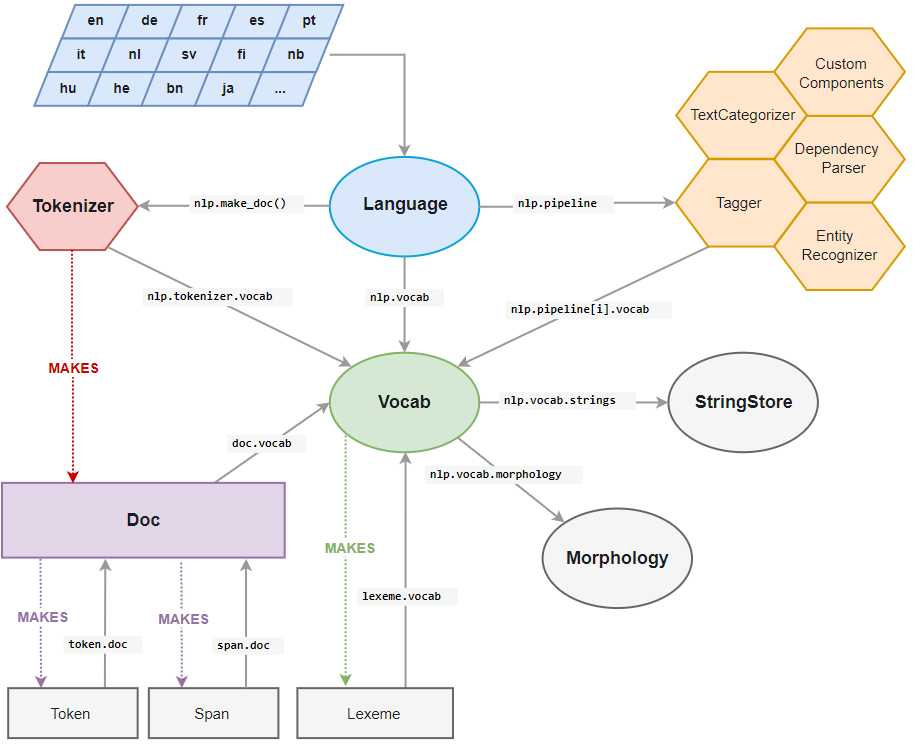
**Fig.5.5 KNN Working(a)**

You intend to find out the class of the blue star (BS). BS can either be RC or GS and nothing else. The “K” is KNN algorithm is the nearest neighbor we wish to take the vote from. Let’s say K = 3. Hence, we will now make a circle with BS as the center just as big as to enclose only three datapoints on the plane. Refer to the following diagram for more details:

[](https://www.analyticsvidhya.com/wp-content/uploads/2014/10/scenario2.png)

**Fig.5.6 KNN Working (b)**

The three closest points to BS is all RC. Hence, with a good confidence level, we can say that the BS should belong to the class RC. Here, the choice became very obvious as all three votes from the closest neighbor went to RC. The choice of the parameter K is very crucial in this algorithm. Next, we will understand what are the factors to be considered to conclude the best K.**import spacy**

****

**Fig.5.7 Spacy Architecture**

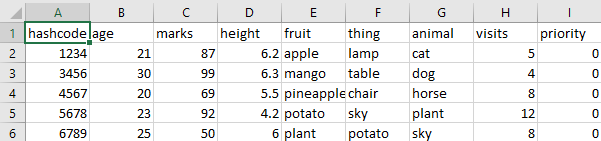
spaCy is an open-source software library for advanced natural language processing, written in the programming languages Python and Cython.

**Nlp = spacy.load(‘en\_core\_web\_md’)**

English multi-task CNN trained on OntoNotes, with GloVe vectors trained on Common Crawl. Assigns word vectors, context-specific token vectors, POS tags, dependency parse and named entities.

* + 1. **Loading data into the data frame and segmenting**

dataset = pd.read\_csv(“C:\\Users\\aweso\\Desktop\\Final year project\\ML\\final.csv”)



**Tab.5.8 Data Set**

The dataset will primarily have 3 main attributes along with other numeric and string data types

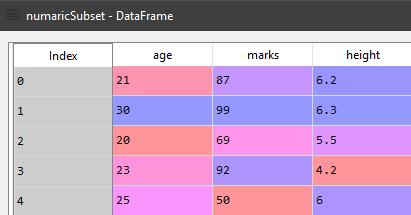
1. Hash-code – represents the identity of the object
2. Visits – represents the number of times that object is visited
3. Priority – the importance of that object for access, initially this value will always remain 0 and will be updated every time the code will run.

**numaricSubset = dataset.select\_dtypes(include = [np.number])**

Stores the attributes with numeric datatype along with the hashcode and the priority attributes.

**numaricSubset = numaricSubset.iloc[:,1:-2]**

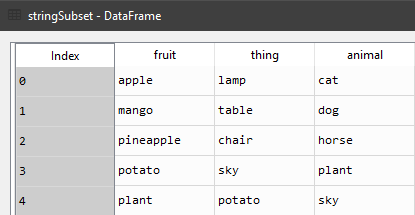
Removes the hashcode and the priority attributes.



**Tab.5.9 Numeric Subset**

**stringSubset = dataset.select\_dtypes(include = [np.object])**

Stores the attributes with String data type



**Tab.5.10 String Subset**

**maxVisitsRowIndex = dataset[“visits”].idxmax()**

Calculates the row with the most number of visits

**maxVisitsRowNumaric = numaricSubset.iloc[maxVisitsRowIndex]**

Stores the numeric values of the entry with the most number of visits.

**maxVisitsRowString = stringSubset.iloc[maxVisitsRowIndex]**

Stores the String values of the entry with the most number of visits

**5.4 CALCULATING THE DISTANCE FOR NUMERIC DATA**

**5.4.1 Function to calculate the distance of numeric attributes**

**def getNumaricDataDistance(numaricSubset:DataFrame , maxVisitsRowNumaric:Series)->Series:**

Here our function takes two parameters, numeric subset data frame which is the dataframe that holds the numeric(integer+Boolean) attributes and the max visited row series that holds all the attributes of the most visited entry.

**Neigh = NearestNeighbors()**

Here we are calling the Nearest Neighbors function and using the neigh variable to call onto its various library unctions.

**Neigh.fit(numaricSubset)**

The fit function is used to fit the numeric subset dataframe as the data to perform the learning upon.

**distanceDetails = neigh.kneighbors([maxVisitsRowNumaric])**

We are passing the max visited row series as a parameter to calculate the distance between the datapoints in our dataframe.

**Distance = distanceDetails[0].flatten()**

**indexes = distanceDetails[1].flatten()**

We flatten the distance and indexes into a 1-D array using the flatten() which returns a copy of the array collapsed into one dimension.

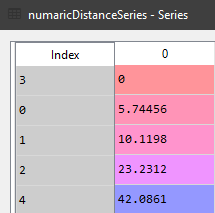
**Return Series(distance , index = indexes)**

We finally return the flattened distance and indexes in form of a series which is a one-dimensional labeled array.

**NumericDistanceSeries= getNumaricDataDistance(numaricSubset,maxVisitsRowNumaric)**

This line is used to call the aforementioned function

To summarize the previous section the getNumaricDataDistance function calculates the Euclidean distance of the most visited entry with respect to the rest of the entries. To achieve this function uses K-NearestNeighbours algorithm to find the required distances.



**Tab.5.11 Numeric Distance**

**5.4.2 Function to calculate the priority of numeric attributes**

**def formatNumaricDistance(numaricDistanceSeries:Series)->Series:**

Here our function takes a single parameternumaricDistanceSeries series that holds all the distance values of the datapoints. The objective of this function is to obtain the priority values ranging between 0-1 which is going to allow us to sort out list based on the calculated priority for faster access times.

**subtractedList = [None]\*len(numaricDistanceSeries)**

Here we make a subtracted list by altering its dimensions with respect to the length of the numeric distance series

**maxValue = numaricDistanceSeries.max()**

We are storing the maximum distance value in the series

**for index in numaricDistanceSeries.index:**

Here we iterate through all the entries and subtract heir index from the max value and store it in the subtracted list index.

**subtractedList[index] = maxValue-numaricDistanceSeries[index]**

**subtractedSeries = Series(subtractedList)**

We typecast the subtracted list to a series.

**subtractedMax = subtractedSeries.max()**

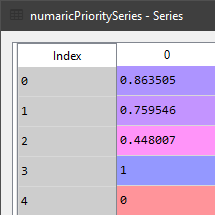
**dividedSeries = subtractedSeries.divide(subtractedMax)**

We finally divide the max valued entry in the subtracted series with the entire series to obtain our priority values.

**Return dividedSeries**

**numaricPrioritySeries = formatNumaricDistance(numaricDistanceSeries)**

We finally use the formatNumaricDistance to calculate the priority series to get the priority series



**Tab.5.12 Numeric Priority**

**5.5 CALCULATING THE DISTANCE FOR STRING DATA**

In order to find the distance between strings we will use an algorithm which finds the similarity between two text and then gives a score between 0 and 1 this result will then be transformed based on the amount of contribution it has to the entire object and then we forward it to priority calculator.

**5.5.1 Function to calculate the similarity of string attributes**

**def findSimilarity(stringSubset:DataFrame,maxVisitsRowString:Series)->DataFrame:**

Here our function takes two parameters string subset Dataframe which is the Dataframe that holds the string attributes and the max visited row series that holds all the attributes of the most visited entry. The objective of this function is to calculate the similarity values of all the string values and store them in a unique dataframe.

**similarity ={}**

Firstly we make an empty list called similarity.

**For (columnName,columnValues) in stringSubset.iteritems():**

We now iterate through all the column names and its respective values in the string subset by making use of the items inbuilt function.

**columnSimilarity = [None]\*len(columnValues)**

Here we make a columnSimilarity list by altering its dimensions with respect to the length of the column values

**for index,item in columnValues.items():**

We further iterate the items in the columnValues with respect to it’s index.

**columnSimilarity[index]=calculateSimilarityScore(item,maxVisitsRowString[columnName])**

Here we’re calling the calculateSimilarityScore function which we shall look into later.

**Similarity[columnName] = columnSimilarity**

**return pd.DataFrame(similarity)**

We copy the similarity scores onto the similarity list and finally store it into a dataframe for further use.

**5.5.2 Function to calculate the similarity score for string attributes**

**def calculateSimilarityScore(word1:str , word2:str)->int:**

This function which is called in the findSimilarity function is used to calculate the similarity between the words passed to the function respectively.

**concatinatedWords = word1+”“+word2**

This concatenates (joins) the two input words.

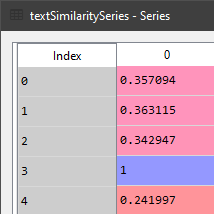
**Tokens = nlp(concatinatedWords)**

Here we use our previously imported en\_core\_web\_md library from spacy to calculate the similarity between the given words.

**Return tokens[0].similarity(tokens[1])**

This finally returns the similarity.

Here we make use of the findSimilarity and the calculateSimilarityScore functions in tandem to calculate the similarity scores of the string data which can further be used in various machine learning algorithms.

****

**Tab.5.13 Text Similarity**

**5.5.3 Function to calculate the average similarity for the string attributes**

**def calcualteAverageSimilarity(similarityDataFrame:DataFrame)->Series:**

This function is used to return a series that contains the average similarity of the similarities between all the entries that was previously calculated in the previous functions. CalcualteAverageSimilarity takes similarityDataFrame that holds the previously calculated similarity scores as the input

**averageSimilarityList = [None]\*len(similarityDataFrame)**

Here we make a averageSimilarity list by altering it’s dimensions with respect to the length of the similarityDataFrame.

**For index,row in similarityDataFrame.iterrows():**

Here we further iterate the items in the similarityDataFrame with respect to it’s index.

**averageSimilarityList[index] = row.mean()**

Now we calculate the mean using the mean() function.

**Return Series(averageSimilarityList)**

**5.5.4 Function to call the similarity functions of the string attributes**

**def getAverageSimilarity(stringSubset:DataFrame,maxVisitsRowString:Series)->Series:**

This is the controller function that calls the above defined functions.

**similarityDataFrame = findSimilarity(stringSubset, maxVisitsRowString)**

This calls the findSimilarity() function.

**averageSimilarity = calcualteAverageSimilarity(similarityDataFrame)**

This calls the calcualteAverageSimilarity () function.

**Return averageSimilarity**

**textSimilaritySeries = getAverageSimilarity(stringSubset,maxVisitsRowString)**

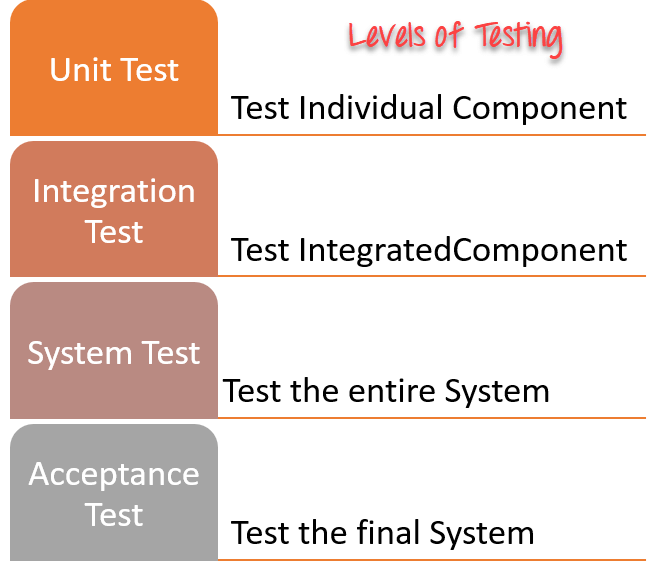
Here we make use of the calcualteAverageSimilarity and the getAverageSimilarity functions in tandem to calculate the average similarity scores from the previously obtained similarity scores.

**CHAPTER 6**

**TESTING AND RESULTS**

**6.1 TESTING**

**Software testing** is an investigation conducted to provide stakeholders with information about the quality of the software product or service under test.[[1]](https://en.wikipedia.org/wiki/Software_testing#cite_note-Kaner_1-1) Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include the process of executing a program or application with the intent of finding software bugs (errors or other defects), and verifying that the software product is fit for use.



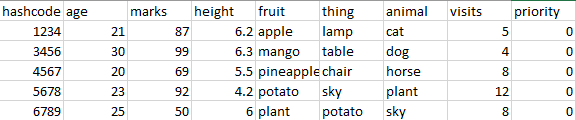
**Fig.6.1 Levels of Testing**

**6.1.1 Unit Testing**

**Unit Testing** is a level of software **testing** where individual units/ components of a software are tested. The purpose is to validate that each **unit** of the software performs as designed. A **unit** is the smallest testable part of any software. It usually has one or a few inputs and usually a single output.

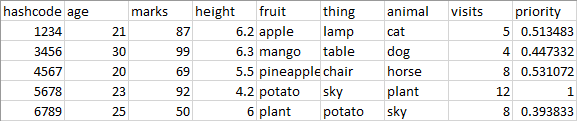
In our project the unit testing consists of testing the modules in complete isolation with each other In order to achieve that we tested some of the components seen below.

**Python api**



**Tab.6.2 Input csv file for machine learning**

This is the input that will be provided to the python api module and the code will be trigger the machine learning code on this csv file and then the priority column will be updated with values as shown below.



**Tab.6.3 Output csv file after machine learning**

We see here that the 4th row has priority 1 because it has most number of visits in the entire dataset and so it has the highest priority and also is the center of calculation for all the machine learning algorithm.

**6.1.2 System Testing**

**SYSTEM TESTING** is a level of **testing** that validates the complete and fully integrated software products. The purpose of a system test is to evaluate the end-to-end system specifications. Usually, the software is only one element of a larger computer-based **system**.

In our project we are testing the entire system by providing entire dataset in form of a student object and then searching for the each object multiple times in order to mimic a typical server side usage of a data structure and then see if the machine learning is working properly or not.

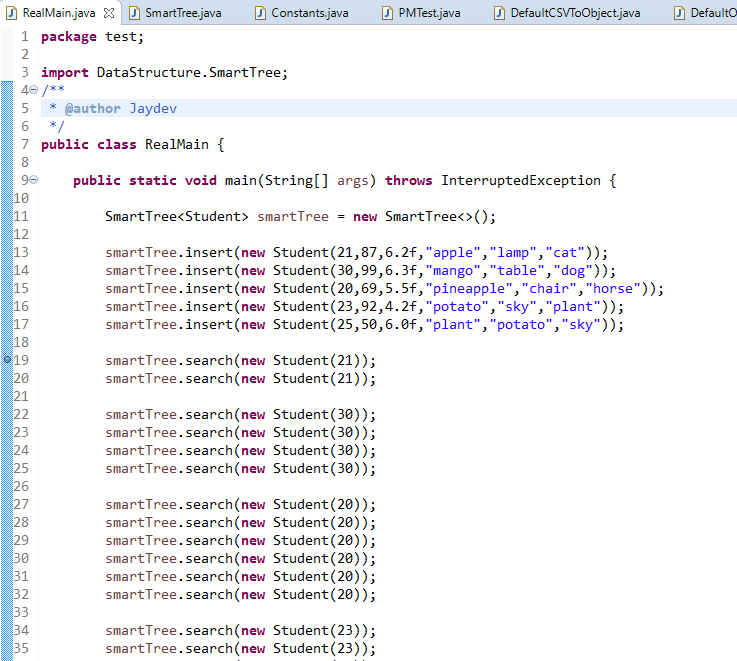
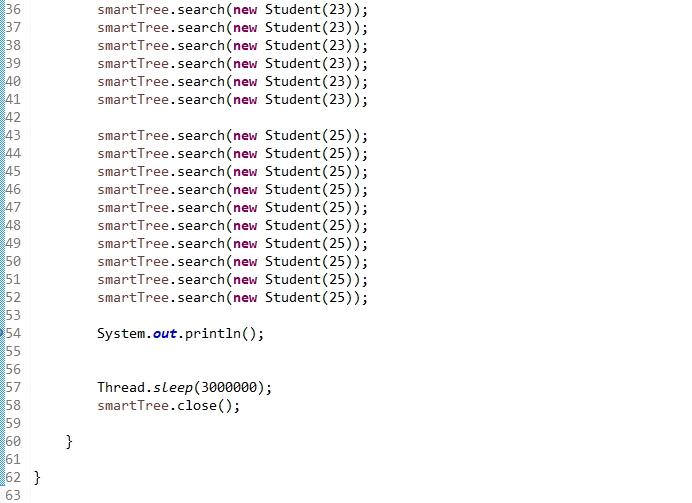


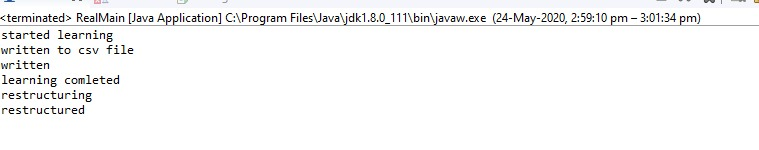
Fig.6.4 System Testing input1



**Fig.6.5 System testing input2**

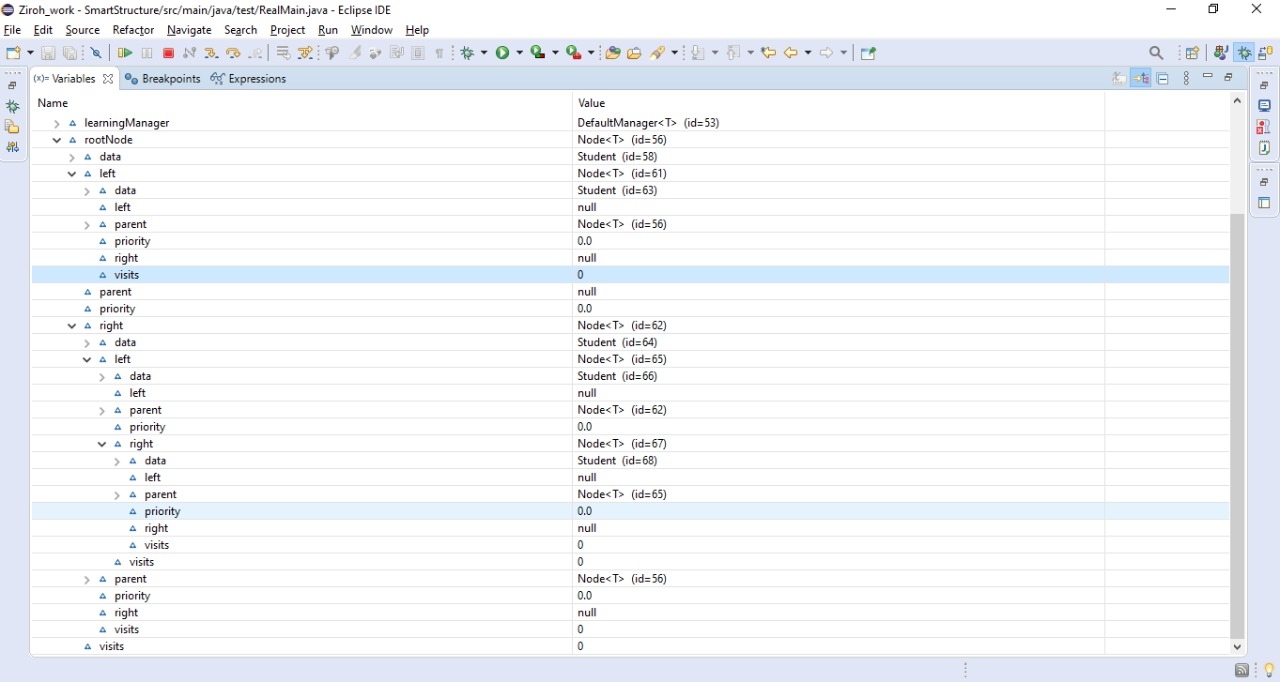
**6.2 RESULTS AND SNAPSHOTS**

Once we execute the above code we get the following result as the output, this is the console screen when testing in order to see if the machine learning is happening properly or not.



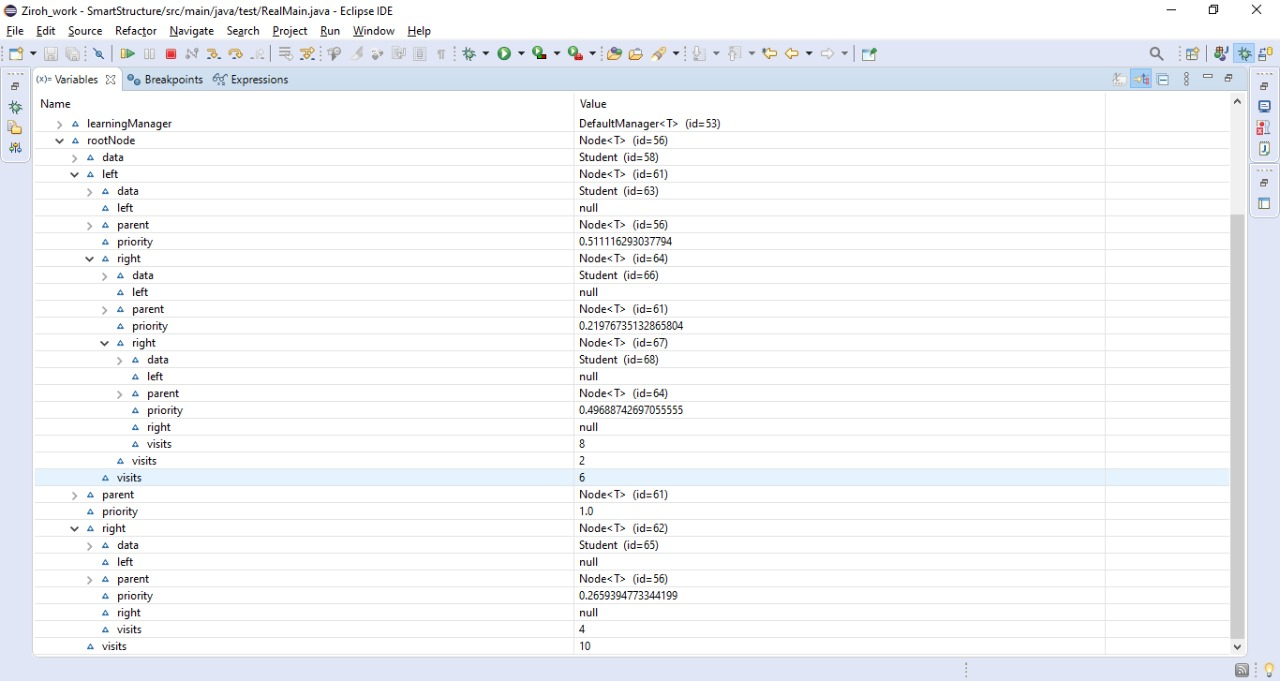
**Fig.6.6 Resultant Output**

In order to validate that the above given code is working internally or not I run the code in debug mode and then look at the variables. Since the tree is represented as references it is easy for us to understand the tree structure with the help of variables shown below. The below representation of the tree shows the tree before rotation we see that the tree is different and the visits and priority are set to null.



**Fig.6.7 Tree before rotation**

Once the machine learning and restructuring takes place the tree structure will change and look like the representation seen below. We see that the priority is set and also the visits are according to the above code.

****

**Fig.6.8 Tree after rotation**

Here there are some points to be noted down -

* root node will always have the priority set to 1 this is because the node with the highest priority are always considered the center when the machine learning takes place and hence the data with the highest visits will always become the root node. This will also be helpful when we see that logically because the data that is visited the most must be at the top this will result that, when further that data will be retrieved it will take O(1) time in order to retrieve it which will be better than that of standard binary search tree where the order would have still been O(h).
* some nodes are no structured properly in the tree as per the priority , this is because the rotation takes place based on leaf nodes one at a time and after one leaf node rotation the other leaf node rotation will affect the previously rotated values , this however will not affect the overall functioning of data.
* In case of a skewed tree this system will also work as a balancer due to the rotation rules of the system, and hence it will make the structure a bit more efficient.

**CHAPTER 7**

**CONCLUSION AND FURTHER WORK**

**CONCLUSION**

The aftermath of our work provides us with a smart and dynamic data retrieval system that can be integrated into any of the existing systems by making use of a java JAR file. The project implements a self-learning smart data structure that uses machine learning to optimize the data retrieval system. This overcomes the need of static data structures that cannot change accordingly with the type of data. By dynamically rotating the binary search tree based on the priorities calculated by the machine learning algorithm that uses a distance based unsupervised learning algorithm, we can significantly improve the time required to access any data on various platforms. Our work gains momentum once it’s integrated with a platform that deals with huge amount of data, as this allows the algorithm to learn better and form relations between the data based on the large database and the huge influx of users. Henceforth our project serves as a dynamic smart data structure that can improve itself based on the data it’s dealing with and significantly improve the data retrieval rate for various types of data.

**FURTHER WORK**

Future improvements to the project may include the use of various different machine learning algorithms that might further improve the ranking of the data in the system. This opens up lots of possibilities in which the smart data structure can be improved and due to its dynamic nature we can even implement it on other existing data structures and improve upon them significantly over time. Another improvement is where we can teach the algorithm to keep track of regular users and find new patterns based on the specific users previous queries, this could significantly improve the speed at which the data in retrieved from the system.

**BIBILOGRAPHY**

[1]Klementiev, Alexandre & Roth, Dan & Small, Kevin. (2008*).* “Unsupervised rank aggregation with distance-based models”. *Proceedings of the 25th International Conference on Machine Learning. 472-479. 10.1145/1390156.1390216*.

[2]Zheng Zhao and Huan Liu. 2007. “Spectral feature selection for supervised and unsupervised learning”. *In Proceedings of the 24th international conference on Machine learning (ICML ’07). Association for Computing Machinery, New York, NY, USA, 1151–1157. DOI:https://doi.org/10.1145/1273496.1273641*

[3]Rajkumar, Arun, and Shivani Agarwal. "A statistical convergence perspective of algorithms for rank aggregation from pairwise data." *International Conference on Machine Learning. 2014.*

[4] R. C. Prati, "Combining feature ranking algorithms through rank aggregation," *The 2012 International Joint Conference on Neural Networks (IJCNN), Brisbane, QLD, 2012, pp. 1-8, doi: 10.1109/IJCNN.2012.6252467.*

[5] Aragon, Cecilia R.; Seidel, Raimund, ["Randomized Search Trees"](http://faculty.washington.edu/aragon/pubs/rst89.pdf) (PDF), [*Proc. 30th Symp. Foundations of Computer Science (FOCS 1989)*](https://en.wikipedia.org/wiki/Symposium_on_Foundations_of_Computer_Science)*, Washington, D.C.: IEEE Computer Society Press, pp. 540–545,*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1109/SFCS.1989.63531*](https://doi.org/10.1109%2FSFCS.1989.63531)*,*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*0-8186-1982-1*](https://en.wikipedia.org/wiki/Special:BookSources/0-8186-1982-1)*,* 1989

[6]Seidel, Raimund; Aragon, Cecilia R. , ["Randomized Search Trees"](http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.30.8602), *Algorithmica,****16****(4/5): 464–497,*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1007/s004539900061*](https://doi.org/10.1007%2Fs004539900061),1996

[7] [Naor, M.](https://en.wikipedia.org/wiki/Moni_Naor); Nissim, K. ,["Certificate revocation and certificate update"](http://eprints.kfupm.edu.sa/29443/1/29443.pdf) (PDF), *IEEE Journal on Selected Areas in Communications,* **18** (4): 561–570, [doi](https://en.wikipedia.org/wiki/Doi_(identifier)):[10.1109/49.839932](https://doi.org/10.1109%2F49.839932),April 2000.

[8] Blelloch, Guy E.; Reid-Miller, Margaret , "Fast set operations using treaps", [*Proc. 10th ACM Symp. Parallel Algorithms and Architectures (SPAA 1998)*](https://en.wikipedia.org/wiki/Symposium_on_Parallel_Algorithms_and_Architectures), New York, NY, USA: ACM, pp. 16–26, [doi](https://en.wikipedia.org/wiki/Doi_(identifier)):[10.1145/277651.277660](https://doi.org/10.1145%2F277651.277660), [ISBN](https://en.wikipedia.org/wiki/ISBN_(identifier)) [0-89791-989-0](https://en.wikipedia.org/wiki/Special:BookSources/0-89791-989-0),1998

[9] [online]https://www.geeksforgeeks.org/construct-bst-from-given-preorder-traversal-set-2/?ref=rp

[10] [online]https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/

[11] [online]https://gameprogrammingpatterns.com/decoupling-patterns.html

[12] [online]https://www.webopedia.com/TERM/D/decoupled.html

[13] [online]https://gameprogrammingpatterns.com/component.html

[14] [online]https://www.iheavy.com/2011/06/16/decoupling-what-is-it-and-why-is-it-important/

[15] [online]https://www.quora.com/What-is-decoupled-architecture-and-how-is-it-implemented

[16][online]https://gameprogrammingpatterns.com/event-queue.html#who-can-write-to-the-queue

**PERFORMA FOR APPROVAL OF THIS PROJECT**

Name of the student :

Name of the Department : Computer Science and Engineering Branch

Year :

Date of Submission :

Batch Number :

|  |  |  |
| --- | --- | --- |
| **Items** | **Yes** | **No** |
| Whether all pages are as per instruction? |  |  |
| Whether all contents of report are in correct order as given in the manual? |  |  |
| Whether the margin specification and spacing for the text correctly followed as per instruction in the manual? |  |  |
| Whether typed in black? |  |  |
| Whether numbers given for figures, tables and other pages serially and in correct position? |  |  |
| Whether tables numbered properly with captions given as top of tables? |  |  |
| Whether figures and diagrams numbered properly with caption given below? |  |  |
| Whether binding is OK? |  |  |
| Whether references are marked in text with square brackets? |  |  |
| Whether CD containing executable s/w, required libraries and source code? |  |  |
| **Comment: Rating: /10** | | |

**Internal Guide**

**Head of the Department**