Me, Varun Nandu and my partner, Suraj Nair are creating a Plagarism Detector for text answers. Thus a particular students answers to various questions can be compared with other students answer and online (Wikipedia) answer to check if there are any similar answers or not. Thus we mentioned in the proposal that we will use FP-growth algorithm to achieve our results of plagiarized work on the following dataset: <https://goo.gl/apF8PA>. We are still going to continue with the same method on the same dataset. The part where we have made progress on is cleaning, arranging of dataset and method of using it.

Just to provide a brief summary of dataset, it consists of answers to various questions of various students. The name of the questions and name of student are provided in each file of dataset. For example : “g4pB\_taska” refers to student “g4pB” and question “taska”. Thus we are using a default dictionary in python to arrange the data in following way:

{{ Question 1 : {Student 1 : Answer}

{Student 2: Answer }

.

.

{Student n: Answer}}

{Question 2 : {Student 1 : Answer}

{Student 2: Answer }

.

.

{Student n: Answer}}

.

. .

}}

Now just to give a brief overview of how our project will work.

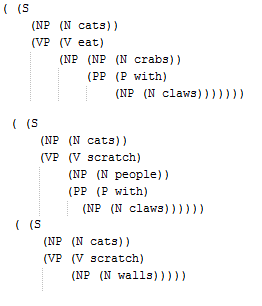
1. The user will select which question he needs to check for plagiarism
2. Based on the input a matrix will be formed as shown in the table below:

|  |  |
| --- | --- |
| Student Name | Answer |
| Student 1 | T11, T12, T13 … T1n |
| Student 2 | T21, T22, T23 … T2n |
| . | . |
| . | . |
| Student n | Tn1, Tn2, Tn3 … Tnn |

The “T11, T21, ….. Tnn” refers to various Trees of the sentences we create based on the students answers. So lets say for student ‘FILL STUDENT NAME SURAJ’ and question ‘QUESTION NAME SURAJ’ first we calculate emission probability and transition probability for each word in the answer. Then we use the Viterbi algorithm for correct tagging of each word in a sentence. Just to share some results, for ‘SURAJ ENTER SENTENCE’ after calculating emission probability and transition probability and applying Viterbi algorithm on it we get the following result:

‘SURAJ ENTER RESULT’.

This is the progress on our project so far. The next steps in our project is to create a CKY tree out of the output of Viterbi algorithm. Thus by learning CFG grammar, our code will create a CKY tree for each sentence which is represented as “T11” in the above table. The first 1 represents the student name and the second 1 represents the first tree. It is important to note that ordering of trees is important here. Once our data is ready we will calculate the support count. Support count is the frequency of occurrence of an item set. In our case the item set will consist of CKY Trees. The trees can be visualized as follows:



Then we apply the FP-growth algorithm which is broken down into two parts, first constructing FP-Tree and second getting frequent item sets. The approach for the same is given below:

Algorithm 1: FP-tree construction

Input: A transaction database DB and a minimum support threshold ?.

Output: FP-tree, the frequent-pattern tree of DB.

Method: The FP-tree is constructed as follows.

1. Scan the transaction database DB once. Collect F, the set of frequent items, and the support of each frequent item. Sort F in support-descending order as FList, the list of frequent items.
2. Create the root of an FP-tree, T, and label it as “null”. For each transaction Trans in DB do the following:

* Select the frequent items in Trans and sort them according to the order of FList. Let the sorted frequent-item list in Trans be [ p | P], where p is the first element and P is the remaining list. Call insert tree([ p | P], T ).
* The function insert tree([ p | P], T ) is performed as follows. If T has a child N such that N.item-name = p.item-name, then increment N ’s count by 1; else create a new node N , with its count initialized to 1, its parent link linked to T , and its node-link linked to the nodes with the same item-name via the node-link structure. If P is nonempty, call insert tree(P, N ) recursively.

Algorithm 2: FP-Growth

Input: A database DB, represented by FP-tree constructed according to Algorithm 1, and a minimum support threshold ?.

Output: The complete set of frequent patterns.

Method: call FP-growth(FP-tree, null).

Procedure FP-growth(Tree, a) {

(01) if Tree contains a single prefix path then { // Mining single prefix-path FP-tree

(02) let P be the single prefix-path part of Tree;

(03) let Q be the multipath part with the top branching node replaced by a null root;

(04) for each combination (denoted as ß) of the nodes in the path P do

(05) generate pattern ß ∪ a with support = minimum support of nodes in ß;

(06) let freq pattern set(P) be the set of patterns so generated;

}

(07) else let Q be Tree;

(08) for each item ai in Q do { // Mining multipath FP-tree

(09) generate pattern ß = ai ∪ a with support = ai .support;

(10) construct ß’s conditional pattern-base and then ß’s conditional FP-tree Tree ß;

(11) if Tree ß ≠ Ø then

(12) call FP-growth(Tree ß , ß);

(13) let freq pattern set(Q) be the set of patterns so generated;

}

(14) return(freq pattern set(P) ∪ freq pattern set(Q) ∪ (freq pattern set(P) × freq pattern set(Q)))

}

Thus when we run the above algorithm we will get frequent item sets which are CKY trees in our case. The minimum support ensures that some sentences of students can be similar. Thus when we will search our dictionary to the trees corresponding to students, we will get students who have copied from each other or Wikipedia. The output of our program is the name of students who have copied for that question.

We will also run our algorithm with lift count and confidence to check how the output changes.

Refrences:

* 1. <http://user.it.uu.se/~kostis/Teaching/DM-05/Slides/association1.pdf>
  2. <https://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Frequent_Pattern_Mining/The_FP-Growth_Algorithm>
  3. <https://www.datasciencecentral.com/profiles/blogs/some-nlp-probabilistic-context-free-grammar-pcfg-and-cky-parsing>
  4. <https://en.wikipedia.org/wiki/Viterbi_algorithm>
  5. An Implementation of the FP-growth Algorithm by Christian Borgelt