NATIONAL UNIVERSITY OF COMPUTING AND EMERGING SCIENCES

ARTIFICIAL INTELLIGENCE

APRIORI ALGORITHM

SECTION: BAI-6A

SUBMITTED BY

ALISHBA SUBHANI
20K-0351

MANNAHIL MIFTAH
20K-0234

SUBMITTED TO

MUHAMMAD DANISH KHAN

PROJECT OVERVIEW

The Apriori algorithm is a popular method for mining frequent itemsets from large datasets. It involves generating candidate itemsets and pruning those that are infrequent. However, as the size of the dataset increases, the computational complexity of Apriori algorithm also increases, making it computationally expensive to execute on a single processor. In this project, we aim to parallelize the Apriori algorithm to improve its scalability and efficiency.

OBJECTIVE

The main objective of this project is to analyze the performance of parallel implementation and compare it with the serial version of the algorithm.

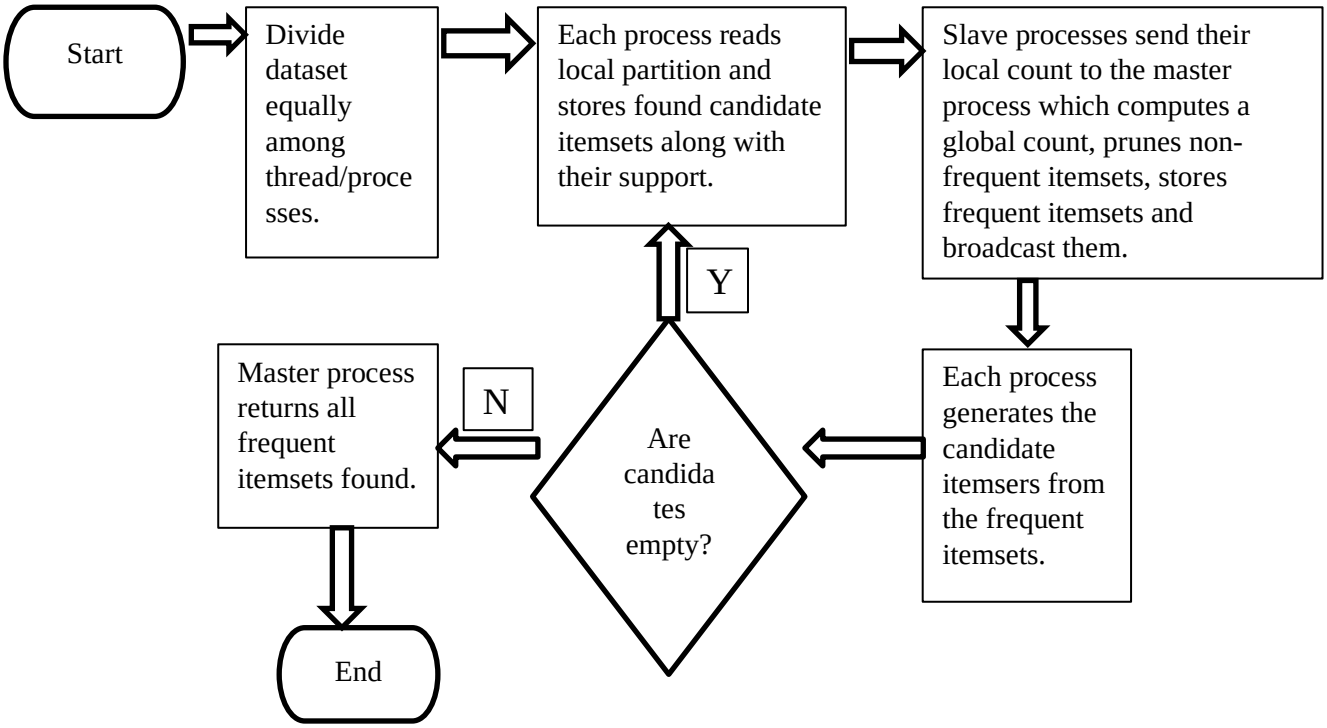
DATASET

The dataset used in this project is taken from Kaggle and contains data of 7,501 transactions.

DESCRIPTION

The parallel implementation of the Apriori algorithm is developed using MPI and OpenMP. The implementation will involve dividing the dataset into smaller subsets and processing them in parallel. The candidate itemsets generated by each subset will be combined and the frequent itemsets will be identified using a global count table. The performance of the parallel implementation will be measured in terms of execution time compared to the serial version of the algorithm.

WORKFLOW


Start
Divide dataset equally among thread/processes.
Each process reads local partition and stores found candidate itemsets along with their support.
Slave processes send their local count to the master process which computes a global count, prunes non-frequent itemsets, stores frequent itemsets and broadcast them.
Each process generates the candidate itemsers from the frequent itemsets.
Are candidates empty?
Y
N
Master process returns all frequent itemsets found.
End

LOCAL RESULTS SAVED IN TEXT FILE

Words each thread got

C:\Users\Mannahil Miftah\Desktop\University\6th semester\PDC\Project\Apriori\Project-Apriori-final\Results.txt - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

Results.txt

```
Thread 0 ---Got Word--- Almonds
Thread 0 ---Got Word--- AntioxydantJuice
Thread 0 ---Got Word--- Asparagus
Thread 0 ---Got Word--- Avocado
Thread 0 ---Got Word--- BabiesFood
Thread 0 ---Got Word--- Bacon
Thread 0 ---Got Word--- BarbecueSauce
Thread 0 ---Got Word--- BlackTea
Thread 0 ---Got Word--- Blueberries
Thread 0 ---Got Word--- BodySpray
Thread 0 ---Got Word--- Bramble
Thread 0 ---Got Word--- Brownies
Thread 0 ---Got Word--- BugSpray
Thread 0 ---Got Word--- BurgerSauce
Thread 3 ---Got Word--- FreshBread
Thread 3 ---Got Word--- FreshTuna
Thread 3 ---Got Word--- FromageBlanc
Thread 3 ---Got Word--- FrozenSmoothie
Thread 3 ---Got Word--- FrozenVegetables
Thread 3 ---Got Word--- GlutenFreeBar
Thread 3 ---Got Word--- GratedCheese
Thread 3 ---Got Word--- GreenBeans
Thread 3 ---Got Word--- GreenGrapes
Thread 3 ---Got Word--- GreenTea
Thread 3 ---Got Word--- GroundBeef
Thread 3 ---Got Word--- Gums
Thread 3 ---Got Word--- Ham
Thread 3 ---Got Word--- HandProteinBar
Thread 3 ---Got Word--- Herb&Pepper
Thread 1 ---Got Word--- Burgers
Thread 1 ---Got Word--- Butter
```

```
Thread 4 ---Got Word--- MintGreenTea
Thread 6 ---Got Word--- Soda
Thread 6 ---Got Word--- Soup
Thread 6 ---Got Word--- Spaghetti
Thread 6 ---Got Word--- SparklingWater
Thread 6 ---Got Word--- Spinach
Thread 6 ---Got Word--- Strawberries
Thread 5 ---Got Word--- Muffins
Thread 5 ---Got Word--- MushroomCreamSauce
Thread 5 ---Got Word--- Napkins
Thread 5 ---Got Word--- NonfatMilk
Thread 5 ---Got Word--- Oatmeal
Thread 5 ---Got Word--- Oil
Thread 5 ---Got Word--- OliveOil
Thread 5 ---Got Word--- Pancakes
Thread 5 ---Got Word--- ParmesanCheese
Thread 5 ---Got Word--- Pasta
Thread 5 ---Got Word--- Pepper
Thread 5 ---Got Word--- PetFood
Thread 5 ---Got Word--- Pickles
Thread 5 ---Got Word--- ProteinBar
Thread 5 ---Got Word--- RedWine
Thread 7 ---Got Word--- VegetablesMix
Thread 7 ---Got Word--- WaterSpray
Thread 7 ---Got Word--- WhiteWine
Thread 7 ---Got Word--- WholeWeatFlour
Thread 7 ---Got Word--- WholeWheatPasta
Thread 7 ---Got Word--- WholeWheatRice
Thread 7 ---Got Word--- Yams
Thread 7 ---Got Word--- YogurtCake
```

Combinations each thread made

Results.txt

```
Thread 2 ---Made Combination--- Eggs EnergyBar
Thread 4 ---Made Combination--- HotDogs LightCream
Thread 6 ---Made Combination--- Pepper ProteinBar
Thread 5 ---Made Combination--- Mint Muffins
Thread 7 ---Made Combination--- TomatoJuice TomatoSauce
Thread 0 ---Made Combination---  Almonds
Thread 1 ---Made Combination--- Carrots Cereals
Thread 3 ---Made Combination--- FrozenSmoothie FrozenVegetables
Thread 2 ---Made Combination--- Eggs EnergyDrink
Thread 4 ---Made Combination--- HotDogs LightMayo
Thread 6 ---Made Combination--- Pepper RedWine
Thread 5 ---Made Combination--- Mint MushroomCreamSauce
Thread 7 ---Made Combination--- TomatoJuice Tomatoes
Thread 1 ---Made Combination--- Carrots Champagne
Thread 3 ---Made Combination--- FrozenSmoothie GratedCheese
Thread 2 ---Made Combination--- Eggs Escalope
Thread 4 ---Made Combination--- HotDogs LowFatYogurt
Thread 6 ---Made Combination--- Pepper Rice
Thread 5 ---Made Combination--- Mint NonfatMilk
Thread 0 ---Made Combination---  Avocado
Thread 7 ---Made Combination--- TomatoJuice Turkey
Thread 1 ---Made Combination--- Carrots Chicken
Thread 3 ---Made Combination--- FrozenSmoothie GreenTea
Thread 2 ---Made Combination--- Eggs ExtraDarkChocolate
Thread 4 ---Made Combination--- HotDogs Magazines
Thread 6 ---Made Combination--- Pepper Salmon
Thread 5 ---Made Combination--- Mint Oil
Thread 7 ---Made Combination--- TomatoJuice VegetablesMix
Thread 1 ---Made Combination--- Carrots Chocolate
Thread 3 ---Made Combination--- FrozenSmoothie GroundBeef
Thread 2 ---Made Combination--- Eggs FrenchFries
```

```
Thread 3 ---Made Combination--- Gums MushroomCreamSauce
Thread 1 ---Made Combination--- Chicken OliveOil
Thread 4 ---Made Combination--- Melons Yams
Thread 2 ---Made Combination--- ExtraDarkChocolate Magazines
Thread 0 ---Made Combination--- Avocado Honey
Thread 3 ---Made Combination--- Gums NonfatMilk
Thread 1 ---Made Combination--- Chicken Pancakes
Thread 4 ---Made Combination--- Melons YogurtCake
Thread 2 ---Made Combination--- ExtraDarkChocolate Meatballs
Thread 0 ---Made Combination--- Avocado HotDogs
Thread 3 ---Made Combination--- Gums Oil
Thread 1 ---Made Combination--- Chicken ParmesanCheese
Thread 4 ---Made Combination--- Milk MineralWater
Thread 2 ---Made Combination--- ExtraDarkChocolate Melons
Thread 0 ---Made Combination--- Avocado LightCream
Thread 3 ---Made Combination--- Gums OliveOil
Thread 1 ---Made Combination--- Chicken Pasta
Thread 4 ---Made Combination--- Milk Mint
Thread 2 ---Made Combination--- ExtraDarkChocolate Milk
Thread 0 ---Made Combination--- Avocado LightMayo
Thread 3 ---Made Combination--- Gums Pancakes
```

Item transactions each thread found

```
Thread 2 ---Finding Items Of Transaction--- Chicken Honey   Muffins YogurtCake
Thread 3 ---Finding Items Of Transaction--- EnergyDrink Escalope    Pancakes
Thread 0 ---Finding Items Of Transaction--- Burgers Eggs    EnergyBar   FrenchWine  Herb&Pepper Meatballs   MineralWater    Olive
Thread 5 ---Finding Items Of Transaction--- Bacon   Burgers Cake    Carrots Chocolate   Cider   Eggs    GreenTea    MineralWater
Thread 7 ---Finding Items Of Transaction--- Soda
Thread 1 ---Finding Items Of Transaction--- Burgers FrenchFries Turkey  Yams
Thread 0 ---Finding Items Of Transaction--- Champagne   LowFatYogurt
Thread 4 ---Finding Items Of Transaction--- Cauliflower EnergyBar   Escalope    FrozenVegetables    GratedCheese    Honey   Olive
Thread 6 ---Finding Items Of Transaction--- BodySpray   FrenchFries LightMayo
Thread 2 ---Finding Items Of Transaction--- BlackTea    Champagne   Chicken Eggs    EnergyBar   LightMayo
Thread 5 ---Finding Items Of Transaction--- Eggs
Thread 3 ---Finding Items Of Transaction--- BugSpray    BurgerSauce Cake    LightCream  MineralWater
Thread 7 ---Finding Items Of Transaction--- Almonds GroundBeef
Thread 0 ---Finding Items Of Transaction--- Champagne
Thread 1 ---Finding Items Of Transaction--- Soda    Spaghetti   Turkey
Thread 4 ---Finding Items Of Transaction--- ExtraDarkChocolate  MineralWater    Soup
Thread 3 ---Finding Items Of Transaction--- Tomatoes    WholeWheatPasta
Thread 6 ---Finding Items Of Transaction--- FrenchFries Melons  MineralWater
Thread 2 ---Finding Items Of Transaction--- GreenTea    LightMayo   OliveOil    Soup    Zucchini
Thread 5 ---Finding Items Of Transaction--- Cake    Eggs    FrenchFries FreshBread  FrozenSmoothie  GreenGrapes Milk
Thread 7 ---Finding Items Of Transaction--- BodySpray   Burgers Eggs    FrozenSmoothie
Thread 0 ---Finding Items Of Transaction--- Almonds Burgers Cookies Eggs    FrenchFries GreenTea
Thread 1 ---Finding Items Of Transaction--- Avocado Cereals Chocolate   EnergyBar   FrozenVegetables    GreenTea    GroundBeef  M
Thread 2 ---Finding Items Of Transaction--- GlutenFreeBar   Muffins
Thread 4 ---Finding Items Of Transaction--- BlackTea    Cake    Honey   RedWine
Thread 3 ---Finding Items Of Transaction--- BabiesFood
Thread 6 ---Finding Items Of Transaction--- Chicken FrenchFries GratedCheese    Herb&Pepper
Thread 5 ---Finding Items Of Transaction--- Cake    Chicken GratedCheese    LowFatYogurt    MineralWater
Thread 7 ---Finding Items Of Transaction--- EnergyBar   Shrimp  Spaghetti
Thread 0 ---Finding Items Of Transaction--- BodySpray   Escalope    Ham Soup
Thread 1 ---Finding Items Of Transaction--- FreshBread  FrozenVegetables    MineralWater    Spaghetti   Tomatoes    WhiteWine   Z
```

Combinations made by each thread

```
Thread 3 ---Made Combination--- Cake FrenchFries Milk MineralWater
Thread 2 ---Made Combination---  Salmon FrozenVegetables MineralWater
Thread 0 ---Made Combination---  Almonds Chicken Milk
Thread 4 ---Made Combination--- Chocolate Pancakes Spaghetti WholeWheatRice
Thread 1 ---Made Combination---  FrozenVegetables Eggs Tomatoes
Thread 6 ---Made Combination--- FrozenVegetables Tomatoes GratedCheese MineralWater
Thread 7 ---Made Combination--- Milk Spaghetti MineralWater Soup
Thread 5 ---Made Combination--- Eggs WholeWheatRice FrozenVegetables Shrimp
Thread 3 ---Made Combination--- Cake FrenchFries Milk OliveOil
Thread 2 ---Made Combination---  Salmon FrozenVegetables OliveOil
Thread 0 ---Made Combination---  Almonds Chicken MineralWater
Thread 4 ---Made Combination--- Chocolate Salmon Chocolate Shrimp
Thread 1 ---Made Combination---  FrozenVegetables Eggs Turkey
Thread 6 ---Made Combination--- FrozenVegetables Tomatoes GratedCheese Spaghetti
Thread 7 ---Made Combination--- Milk Spaghetti MineralWater Spaghetti
Thread 5 ---Made Combination--- Eggs WholeWheatRice FrozenVegetables Spaghetti
Thread 3 ---Made Combination--- Cake FrenchFries Milk Pancakes
Thread 2 ---Made Combination---  Salmon FrozenVegetables Pancakes
Thread 4 ---Made Combination--- Chocolate Salmon Chocolate Soup
Thread 1 ---Made Combination---  FrozenVegetables Eggs WholeWheatRice
Thread 6 ---Made Combination--- FrozenVegetables Tomatoes GreenTea GroundBeef
Thread 7 ---Made Combination--- Milk Spaghetti MineralWater Tomatoes
Thread 5 ---Made Combination--- Eggs WholeWheatRice FrozenVegetables Tomatoes
Thread 3 ---Made Combination--- Cake FrenchFries Milk Shrimp
Thread 0 ---Made Combination---  Almonds Chicken Spaghetti
Thread 2 ---Made Combination---  Salmon FrozenVegetables Shrimp
Thread 4 ---Made Combination--- Chocolate Salmon Chocolate Spaghetti
Thread 1 ---Made Combination---  FrozenVegetables Escalope FrenchFries
Thread 6 ---Made Combination--- FrozenVegetables Tomatoes GreenTea Milk
Thread 7 ---Made Combination--- Milk Spaghetti MineralWater Turkey
Thread 5 ---Made Combination--- Eggs WholeWheatRice GratedCheese GroundBeef
```

GLOBAL RESULT (FREQUENT ITEMS)

- *OPENMP*

```
C:\Users\Mannahil Miftah\De:

FREQUENT ITEMS ARE
Chocolate Soup
FrozenVegetables LowFatYogurt
Cake FrozenVegetables
Cereals MineralWater
RedWine Spaghetti
Chocolate Cookies
FrenchFries GratedCheese
Burgers FrozenVegetables
Burgers Pancakes
Cookies Eggs
FrenchFries WholeWheatRice
Pancakes Shrimp
Burgers Turkey
FrenchFries Turkey
Chocolate Salmon
OliveOil Pancakes
Chocolate GratedCheese
Eggs WholeWheatRice
MineralWater RedWine
Chicken FrenchFries
Eggs Escalope
Eggs FrozenSmoothie
FrozenSmoothie GreenTea
Shrimp Tomatoes
Chocolate Turkey
FrozenVegetables OliveOil
GratedCheese GroundBeef
Milk Turkey
Burgers Cake
CookingOil Milk
GreenTea Shrimp
GroundBeef Shrimp
Avocado MineralWater
Champagne Chocolate
CookingOil Eggs
GroundBeef Tomatoes
Cake Pancakes
Chicken GreenTea
```

```
C:\Users\Mannahil Miftah\De:

GreenTea Turkey
Honey Spaghetti
Milk WholeWheatRice
Burgers GroundBeef
Chocolate WholeWheatRice
Cookies GreenTea
Eggs OliveOil
FrenchFries Tomatoes
Eggs Tomatoes
GreenTea Tomatoes
Eggs Herb&Pepper
LowFatYogurt Milk
Cake Milk
Cookies FrenchFries
FrenchFries LowFatYogurt
FreshBread MineralWater
FrozenVegetables Pancakes
Salmon Spaghetti
Cake Chocolate
Chocolate CookingOil
FrenchFries GroundBeef
Chocolate Tomatoes
Escalope Spaghetti
Milk Tomatoes
Cake GreenTea
Eggs Shrimp
GroundBeef OliveOil
Spaghetti WholeWheatRice
FrozenSmoothie Milk
Soup Spaghetti
Chicken Eggs
FrozenVegetables GreenTea
FrenchFries FrozenSmoothie
GroundBeef Pancakes
Chicken Chocolate
Chicken Milk
Chocolate LowFatYogurt
GreenTea GroundBeef
Chocolate FrozenSmoothie
Honey MineralWater
```

```
C:\Users\Mannahil Miftah\De:
LowFatYogurt Spaghetti
Milk Soup
FrozenSmoothie Spaghetti
CookingOil Spaghetti
GroundBeef Herb&Pepper
FrozenVegetables Tomatoes
Herb&Pepper Spaghetti
Chocolate OliveOil
Escalope FrenchFries
GreenTea Pancakes
GratedCheese Spaghetti
Milk Pancakes
Spaghetti Turkey
FrozenVegetables Shrimp
Eggs LowFatYogurt
FrozenVegetables GroundBeef
Burgers Chocolate
Escalope MineralWater
Herb&Pepper MineralWater
Milk OliveOil
MineralWater Salmon
Chicken Spaghetti
Burgers GreenTea
GratedCheese MineralWater
Chocolate Escalope
GreenTea Milk
Milk Shrimp
Burgers Milk
Cake FrenchFries
Chocolate Shrimp
Cake Spaghetti
Cake Eggs
FrenchFries FrozenVegetables
MineralWater Turkey
Eggs Turkey
Chocolate Pancakes
Eggs GroundBeef
CookingOil MineralWater
FrenchFries Pancakes
MineralWater WholeWheatRice
```

```
C:\Users\Mannahil Miftah\De:
FrenchFries Milk
LowFatYogurt MineralWater
Burgers MineralWater
MineralWater Tomatoes
Pancakes Spaghetti
Eggs GreenTea
GreenTea Spaghetti
Cake MineralWater
FrenchFries Spaghetti
MineralWater OliveOil
FrozenVegetables Spaghetti
FrenchFries GreenTea
Burgers Eggs
Eggs Milk
GreenTea MineralWater
Chocolate Milk
Chocolate Eggs
FrenchFries MineralWater
MineralWater Pancakes
Chocolate FrenchFries
Milk Spaghetti
FrozenVegetables MineralWater
Eggs FrenchFries
Eggs Spaghetti
GroundBeef Spaghetti
Chocolate Spaghetti
GroundBeef MineralWater
Milk MineralWater
Eggs MineralWater
Chocolate MineralWater
MineralWater Spaghetti

--------------------------------
Process exited after 196.9 seconds with return value 0
Press any key to continue . . .
```

- *MPI*

```
Time passed: 14.7752
KEY      VALUE
almonds 0.0203973
avocado 0.0333289
avocado mineral water    0.0115985
barbecue sauce   0.0107986
black tea        0.0142648
body spray       0.0114651
brownies         0.0337288
burgers 0.0871884
burgers cake     0.0114651
burgers chocolate        0.0170644
burgers eggs     0.0287962
burgers french fries     0.0219971
burgers frozen vegetables        0.0105319
burgers green tea        0.0174643
burgers ground beef      0.0119984
burgers milk     0.0178643
burgers mineral water    0.0243967
burgers pancakes         0.0105319
burgers spaghetti        0.0214638
burgers turkey   0.0106652
butter   0.0301293
cake     0.0810559
cake chocolate   0.0135982
cake eggs        0.0190641
cake french fries        0.0178643
cake frozen vegetables   0.0102653
cake green tea   0.0141314
cake milk        0.0133316
cake mineral water       0.027463
cake pancakes    0.0118651
cake spaghetti   0.0181309
carrots 0.0153313
cereals 0.0257299
cereals mineral water    0.0102653
champagne        0.0467938
champagne chocolate      0.0115985
chicken 0.059992
chicken chocolate        0.014798
chicken eggs     0.0143981
chicken french fries     0.0110652
chicken green tea        0.0118651
chicken milk     0.014798
chicken mineral water    0.022797
chicken spaghetti        0.0171977
chocolate        0.163978
chocolate cookies        0.0103986
```

RESULTS OF SERIAL IMPLEMENTATION

```
C:\Users\Mannahil Miftah\De

Read
Time passed: 24.3578


FREQUENT ITEMS ARE
Eggs MineralWater
Chocolate MineralWater
MineralWater Spaghetti

--------------------------------
Process exited after 30.14 seconds with return value 0
Press any key to continue . . .
```

CODE

OPENMP

```
#include <sstream>
#include <map>
#include <algorithm>
#include <sys/time.h>
using namespace std;

void read_file(char file_name[], vector< vector<string> > &matrix, map<string,float>
&dictionary);
void find_itemsets(vector<string> matrix, vector<string> candidates, map<string,float>
&temp_dictionary, int k, int item_idx, string itemset, int current, vector<string>
single_candidates);
void prune_itemsets(map<string,float> &temp_dictionary, vector<string> &candidates, float
min_support, vector<string> &single_candidates);
void update_candidates(vector<string> &candidates, vector<string> freq_itemsets,
vector<string> &single_candidates);

// ------------------------------------------------------------
// Main
// ------------------------------------------------------------
```

```
int main(){
ofstream outfile;
    outfile.open("Results.txt");
    char* file_name = "project.csv";
    float min_support = .01;
    vector< vector<string> > matrix;
    map<string,float> dictionary;
    map<string,float> temp_dictionary;
    vector<string> candidates;
    vector<string> single_candidates;
    vector<vector<string>> gotWords;
    vector<string> transaction;
    vector<string> combination;
    int n_rows;
    string item;

    cout<<"Max threads: "<<omp_get_max_threads()<<endl;

    struct timeval start, end;
    double elapsed;

    gettimeofday(&start, NULL);

    // read file into 2D vector matrix and insert 1-itemsets in dictionary as key with their frequency
as value
    read_file(file_name, matrix, dictionary);

    n_rows = matrix.size();
vector<string> gotword;
    // divide frequency by number of rows to calculate support
    #pragma omp parallel for ordered
    for (int i=0; i<dictionary.size(); i++) {
        map<string, float>::iterator itr = dictionary.begin();
        advance(itr, i);
        itr->second = itr->second/float(n_rows);
        sleep(.1);
        #pragma omp critical
        {
            cout << "Thread " << omp_get_thread_num() << " ---Got Word--- " << itr->first << endl;
```

```
            outfile << "Thread " << omp_get_thread_num() << " ---Got Word--- " << itr->first <<
endl;
}
    }
outfile.close();
    // prune from dictionary 1-itemsets with support < min_support and insert items in candidates
vector
    prune_itemsets(dictionary, candidates, min_support, single_candidates);

    // insert in dictionary all k-itemset
    int n = 2; // starting from 2-itemset
    while(!candidates.empty()){

        temp_dictionary.clear();
        // read matrix and insert n-itemsets in temp_dictionary as key with their frequency as value
        #pragma omp parallel for
        for (int i = 0; i < matrix.size(); i++){


            #pragma omp critical
{
ofstream outfile;
                                outfile.open("Results.txt", ios::app);
            cout << "Thread " << omp_get_thread_num() << " ---Finding Items Of Transaction--- ";
            outfile << "Thread " << omp_get_thread_num() << " ---Finding Items Of Transaction---
";
            for(auto j:matrix[i]){
if(j.empty())
continue;
else
                                cout << j << "\t";
                                outfile << j << "\t";
            }
            outfile << endl;
                        cout << endl;
            }


            find_itemsets(matrix[i], candidates, temp_dictionary, n, -1, "", 0, single_candidates);
        }
```

```
        outfile.close();
        // divide frequency by number of rows to calculate support
        #pragma omp parallel for
        for (int i=0; i<temp_dictionary.size(); i++) {
            map<string, float>::iterator itr = temp_dictionary.begin();
            advance(itr, i);
            itr->second = itr->second/float(n_rows);
        }
        // prune from temp_dictionary n-itemsets with support < min_support and insert items in
candidates vector
        prune_itemsets(temp_dictionary, candidates, min_support, single_candidates);
        // append new n-itemsets to main dictionary
        dictionary.insert(temp_dictionary.begin(), temp_dictionary.end());
        n++;
    }

    gettimeofday(&end, NULL);
    elapsed = (end.tv_sec - start.tv_sec) +
              ((end.tv_usec - start.tv_usec)/1000000.0);
    cout<<"Time passed: "<<elapsed<<endl;

    //removing extra space at the beginning of the word
    for ( auto it = dictionary.begin(); it != dictionary.end(); ) {

            //const_cast is used to change the constant value of any object
        string& key = const_cast<string&>(it->first);
        size_t pos = key.find_first_not_of(' ');
        //npos indicates no match was found
        if (pos != string::npos) {
            key = key.substr(pos);
            it++;
        } else {
            it = dictionary.erase(it);
        }
    }

    // to sort the items on the basis of support value
    multimap<float,string> sorted_map;
    for (auto& it : dictionary){
            sorted_map.insert({it.second, it.first});
```

```
}

cout << endl << endl;
cout << "FREQUENT ITEMS ARE\n";

//printing the most frequent items
for (const auto& entry : sorted_map) {
    const float& value = entry.first;
    const string& key = entry.second;

    if (key.find(" ") != string::npos) {
        cout << key << "\t" << value << endl;
    }
}

    return 0;
}


void read_file(char file_name[], vector< vector<string> > &matrix, map<string,float>
&dictionary){
    ifstream myfile (file_name);

    vector<string> row;
    string line;
    stringstream ss;
    string item;

    while(getline (myfile, line)){
        ss << line;

        while(getline (ss, item, ',')) {
            item.erase(remove(item.begin(), item.end(), '\r'), item.end());
            row.push_back(item);
            // insert item into dictionary and increment its value
            dictionary[item]++;
        }

        sort(row.begin(), row.end());
        matrix.push_back(row);
```

```
        ss.clear();
        row.clear();
    }

    myfile.close();
}

void find_itemsets(vector<string> matrix, vector<string> candidates, map<string,float>
&temp_dictionary, int k, int item_idx, string itemset, int current, vector<string>
single_candidates){
    if(current == k){
        itemset = itemset.erase(0,1); // remove first space

        // if itemset is a candidate insert it into temp_dictionary to calculate support
        if(find(candidates.begin(), candidates.end(), itemset) != candidates.end()){

            #pragma omp critical
            temp_dictionary[itemset]++;
            return;
        }
        // if itemset is not a candidate discard it
        else{
            return;
        }
    }

    string item;
    for (int j = ++item_idx; j < matrix.size(); j++){
        item = matrix[j];

        // if item does not compose one of the candidates, skip it
        if(!(find(single_candidates.begin(), single_candidates.end(), item) !=
single_candidates.end())){
            continue;
        }

        find_itemsets(matrix, candidates, temp_dictionary, k, j, itemset + " " + item, current+1,
single_candidates);
    }
```

```
}

void prune_itemsets(map<string,float> &temp_dictionary, vector<string> &candidates, float
min_support, vector<string> &single_candidates){
    vector<string> freq_itemsets;
    candidates.clear(); // empty candidates to then update it
    single_candidates.clear();

    for (map<string, float>::iterator it = temp_dictionary.begin(); it != temp_dictionary.end(); ){ //
like a while
        if (it->second < min_support){
            temp_dictionary.erase(it++);
        }
        else{
            freq_itemsets.push_back(it->first);
            ++it;
        }
    }

    if(!freq_itemsets.empty()){
        update_candidates(candidates, freq_itemsets, single_candidates);
    }
}

void update_candidates(vector<string> &candidates, vector<string> freq_itemsets,
vector<string> &single_candidates){
    string item;
    stringstream to_combine;
    vector<string> items;
    vector<string> elements;
    string combination;

    int common_items;

    #pragma omp parallel for private(item, to_combine, items, elements, combination,
common_items)
    for(int i = 0; i < freq_itemsets.size()-1; i++){
        for(int j = i+1; j < freq_itemsets.size(); j++){
            common_items = 0;
            items.clear();
```

```
        to_combine.clear();
        combination.clear();

        to_combine << freq_itemsets[i] + ',' + freq_itemsets[j];
        while(getline (to_combine, item, ',')) {
            if(!(find(items.begin(), items.end(), item) != items.end())){
                items.push_back(item);
                combination += " " + item;
            }
            else{
                common_items++;
            }
        }

        // if the statement is true than we can add combination as candidate
        // else we created all correct combinations and we pass to the next itemset

if(common_items == items.size()-2){
ofstream outfile;
                        outfile.open("Results.txt", ios::app);
            combination.erase(0,1); // remove first space

            #pragma omp critical
            {

                candidates.push_back(combination);
                cout << "Thread " << omp_get_thread_num() << " ---Made Combination--- " <<
combination << endl;
                outfile << "Thread " << omp_get_thread_num() << " ---Made Combination--- " <<
combination << endl;

                // insert single items candidates
                for(int i=0; i<items.size(); i++) {
                    if(!(find(single_candidates.begin(), single_candidates.end(), items[i]) !=
single_candidates.end())){
                        single_candidates.push_back(items[i]);
                    }
                }

            }
```

```
            outfile.close();
        }

        else{
            break;
        }

    }
  }
}
```

MPI

```
#include <mpi.h>
#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <sstream>
#include <map>
#include <algorithm>
#include <sys/time.h>
using namespace std;

int count_file_lines(char file_name[]);
void compute_local_start_end(char file_name[], int my_rank, int comm_sz, int *local_start, int
*local_end);
void read_file(char file_name[], int local_start, int local_end, vector< vector<string> > &matrix,
map<string,float> &dictionary);
void find_itemsets(vector<string> matrix, vector<string> candidates, map<string,float>
&temp_dictionary, int k, int item_idx, string itemset, int current, vector<string>
single_candidates, int my_rank);
void prune_itemsets_MPI(map<string,float> &temp_dictionary, vector<string> &candidates,
float min_support, int my_rank, int comm_sz, vector<string> &single_candidates);
void broadcast_freq_itemsets(vector<string> &freq_itemsets, int my_rank);
void update_candidates(vector<string> &candidates, vector<string> freq_itemsets,
vector<string> &single_candidates, int rank);

int main(int argc, char** argv){
    MPI_Init(NULL, NULL);
```

```
    int comm_sz;
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
    int my_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    char file_name[] = "project.csv";
    float min_support = 0.01;
    vector< vector<string> > matrix;
    map<string,float> dictionary;
    map<string,float> temp_dictionary;
    vector<string> candidates;
    vector<string> single_candidates;
    int tot_lines;
    int local_start = 0, local_end = 0;
    string item;
    struct timeval start, end;
    double elapsed;
    gettimeofday(&start, NULL);
    compute_local_start_end(file_name, my_rank, comm_sz, &local_start, &local_end);

    // read file into 2D vector matrix and insert 1-itemsets in dictionary as key with their frequency
as value
    read_file(file_name, local_start, local_end, matrix, dictionary);
    tot_lines = count_file_lines(file_name);
cout << "Number of Lines read by Process " << my_rank << ": " << tot_lines << endl;

    // divide frequency by number of rows to calculate support
    for (map<string, float>::iterator i = dictionary.begin(); i != dictionary.end(); ++i) {
        i->second = i->second/float(tot_lines);
    }

    // prune from dictionary 1-itemsets with support < min_support and insert items in candidates
vector
    prune_itemsets_MPI(dictionary, candidates, min_support, my_rank, comm_sz,
single_candidates);
for(int i = 0; i < comm_sz; i++){
if(my_rank == i){
for(auto j: candidates){
cout << "Process " << my_rank << " ---inserted item--- " << j << endl;
}
}
```

```
}

    // insert in dictionary all k-itemset
    int n = 2; // starting from 2-itemset
    while(!candidates.empty()){
        temp_dictionary.clear();
        // read matrix and insert n-itemsets in temp_dictionary as key with their frequency as value
        for (int i = 0; i < matrix.size(); i++){
            find_itemsets(matrix[i], candidates, temp_dictionary, n, -1, "", 0, single_candidates,
my_rank);
        }

        // divide frequency by number of rows to calculate support
        for (map<string, float>::iterator i = temp_dictionary.begin(); i != temp_dictionary.end();
++i) {
            i->second = i->second/float(tot_lines);
        }

        // prune from temp_dictionary n-itemsets with support < min_support and insert items in
candidates vector
        prune_itemsets_MPI(temp_dictionary, candidates, min_support, my_rank, comm_sz,
single_candidates);
        for(auto i:candidates){
            cout << "Process " << my_rank << " ---found frequent item --- " << i << endl;
}

        // append new n-itemsets to main dictionary
        if(my_rank == 0){
            dictionary.insert(temp_dictionary.begin(), temp_dictionary.end());
            for(auto i:temp_dictionary){
            cout << "Process 0 inserted item " << i.first << " with count " << i.second << endl;
}
        }
        n++;
    }
    if(my_rank == 0){
        gettimeofday(&end, NULL);
        elapsed = (end.tv_sec - start.tv_sec) + ((end.tv_usec - start.tv_usec)/1000000.0);
        cout<<"Time passed: "<<elapsed<<endl;
```

```
        //removing extra space at the beginning of the word
    for ( auto it = dictionary.begin(); it != dictionary.end(); ) {
            //const_cast is used to change the constant value of any object
        string& key = const_cast<string&>(it->first);
        size_t pos = key.find_first_not_of(' ');
        if (pos != string::npos) {
            key = key.substr(pos);
            it++;
        } else {
            it = dictionary.erase(it);
        }
    }

    // to sort the items on the basis of support value
    multimap<float,string> sorted_map;
    for (auto& it : dictionary){
            sorted_map.insert({it.second, it.first});
}
cout << endl << endl;
cout << "FREQUENT ITEMS ARE\n";

//printing the most frequent items
for (const auto& entry : sorted_map) {
    const float& value = entry.first;
    const string& key = entry.second;

    if (key.find(" ") != string::npos) {
        cout << key << endl;
    }
}

    }
    MPI_Finalize();
    return 0;
}

int count_file_lines(char file_name[]){
    int tot_lines = 0;=
    string line;
    ifstream myfile (file_name);
```

```
    while(getline (myfile, line)){
        tot_lines++;
    }
    myfile.close();
    return tot_lines;
}

void compute_local_start_end(char file_name[], int my_rank, int comm_sz, int *local_start, int
*local_end){
    int tot_lines;
    int lines_per_each;
    *local_start = 0;
    *local_end = 0;
    tot_lines = count_file_lines(file_name);
    lines_per_each = tot_lines/comm_sz;
    if(tot_lines%comm_sz != 0){
        int zp = comm_sz - (tot_lines % comm_sz);
        for(int i=0; i<my_rank; i++){
            if(i >= zp){
                *local_start += lines_per_each + 1;
            }
            else{
                *local_start += lines_per_each;
            }
        }
        if(my_rank >= zp){
            lines_per_each++;
        }
        *local_end = *local_start + lines_per_each;
    }
    else{
        *local_start = lines_per_each*my_rank;
        *local_end = *local_start + lines_per_each;
    }
}

void read_file(char file_name[], int local_start, int local_end, vector<vector<string>> &matrix,
map<string,float> &dictionary){
    int line_index = 0;
    ifstream myfile (file_name);
```

```
    vector<string> row;
    string line;
    stringstream ss;
    string item;
    while(getline (myfile, line)){
        if(line_index >= local_start & line_index < local_end){
            ss << line;
            while(getline (ss, item, ',')) {
                item.erase(remove(item.begin(), item.end(), '\r'), item.end());
                row.push_back(item);
                // insert item into dictionary and increment its value
                dictionary[item]++;
            }
            sort(row.begin(), row.end());
            matrix.push_back(row);
            ss.clear();
            row.clear();
        }
        if(line_index >= local_end) break;
        line_index++;
    }
    myfile.close();
}

void find_itemsets(vector<string> matrix, vector<string> candidates, map<string,float>
&temp_dictionary, int k, int item_idx, string itemset, int current, vector<string>
single_candidates, int rank){
    if(current == k){
        itemset = itemset.erase(0,1); // remove first space
cout << "Process " << rank << " ---got itemset--- " << itemset << endl;
        // if itemset is a candidate insert it into temp_dictionary to calculate support
        if(find(candidates.begin(), candidates.end(), itemset) != candidates.end()){
            temp_dictionary[itemset]++;
            return;
        }
        // if itemset is not a candidate discard it
        else{
            return;
        }
    }
```

```
    string item;
    for (int j = ++item_idx; j < matrix.size(); j++){
        item = matrix[j];
        // if item does not compose one of the candidates, skip it
        if(!(find(single_candidates.begin(), single_candidates.end(), item) !=
single_candidates.end())){
            continue;
        }
//cout << "Process " << rank << " ---found frequent item--- " << item << endl;
        find_itemsets(matrix, candidates, temp_dictionary, k, j, itemset + " " + item, current+1,
single_candidates, rank);
    }
}

void prune_itemsets_MPI(map<string,float> &temp_dictionary, vector<string> &candidates,
float min_support, int my_rank, int comm_sz, vector<string> &single_candidates){
    string itemsets;
    string item;
    int count;
    vector<float> supports;
    stringstream ss;
    vector<string> freq_itemsets;
    if(my_rank != 0){
        for (map<string, float>::iterator i = temp_dictionary.begin(); i != temp_dictionary.end();
++i) {
            itemsets.append('|' + i->first);
            supports.push_back(i->second);
        }
        itemsets.erase(0,1); // remove first char
        MPI_Send(&itemsets[0], itemsets.length()+1, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
        MPI_Send(&supports[0], supports.size(), MPI_FLOAT, 0, 0, MPI_COMM_WORLD);
    }
    else{
        for(int i=1; i<comm_sz; i++){
            MPI_Status status;
            MPI_Probe(i, 0, MPI_COMM_WORLD, &status);
            MPI_Get_count(&status, MPI_CHAR, &count);
            char buf[count];
            MPI_Recv(&buf, count, MPI_CHAR, i, 0, MPI_COMM_WORLD, &status);
            itemsets = buf;
```

```
            MPI_Probe(i, 0, MPI_COMM_WORLD, &status);
            MPI_Get_count(&status, MPI_FLOAT, &count);
            supports.resize(count);
            MPI_Recv(&supports[0], count, MPI_FLOAT, i, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            ss << itemsets;
            count = 0;
            while(getline (ss, item, '|')) {
                temp_dictionary[item] += supports[count];
                count++;
            }
            supports.clear();
            itemsets.clear();
            ss.clear();
        }
        // prune itemsets to obtain just frequent ones
        for (map<string, float>::iterator it = temp_dictionary.begin(); it != temp_dictionary.end(); ){
// like a while
            if (it->second < min_support){
                temp_dictionary.erase(it++);
            }
            else{
                freq_itemsets.push_back(it->first);
                ++it;
            }
        }
    }
    broadcast_freq_itemsets(freq_itemsets, my_rank);
    candidates.clear(); // empty candidates to then update it
    single_candidates.clear();
    if(!freq_itemsets.empty()){
        update_candidates(candidates, freq_itemsets, single_candidates, my_rank);
    }
}

void broadcast_freq_itemsets(vector<string> &freq_itemsets, int my_rank){
    char* buf;
    string itemsets;
    string item;
    int count;
```

```
    stringstream ss;
    if(my_rank == 0){
        for(int i=0; i<freq_itemsets.size(); i++) {
            itemsets.append('|' + freq_itemsets[i]);
        }
        itemsets.erase(0,1); // remove first char
        count = itemsets.length()+1;
    }
    MPI_Bcast(&count, 1, MPI_INT, 0, MPI_COMM_WORLD);
    if(my_rank == 0){
        buf = (char*)malloc(sizeof(char)*count);
        strcpy(buf, itemsets.c_str());
    }
    else{
        buf = (char*)malloc(sizeof(char)*count);
    }
    MPI_Bcast(&buf[0], count, MPI_CHAR, 0, MPI_COMM_WORLD);
    if(my_rank != 0){
        itemsets = buf;
        ss << itemsets;
        while(getline (ss, item, '|')) {
            freq_itemsets.push_back(item);
        }
    }
}

void update_candidates(vector<string> &candidates, vector<string> freq_itemsets,
vector<string> &single_candidates, int rank){
    string item;
    stringstream to_combine;
    vector<string> items;
    vector<string> elements;
    string combination;
    int common_items;
    for(int i = 0; i < freq_itemsets.size()-1; i++){
        for(int j = i+1; j < freq_itemsets.size(); j++){
            common_items = 0;
            items.clear();
            to_combine.clear();
            combination.clear();
```

```
            to_combine << freq_itemsets[i] + ',' + freq_itemsets[j];
            while(getline (to_combine, item, ',')) {
                if(!(find(items.begin(), items.end(), item) != items.end())){
                    items.push_back(item);
                    combination += " " + item;
                }
                else{
                    common_items++;
                }
            }
cout << "Process " << rank << "---found combination--- " << combination << endl;
            // if the statement is true than we can add combination as candidate
            // else we created all correct combinations and we pass to the next itemset
            if(common_items == items.size()-2){
                combination.erase(0,1); // remove first space
                candidates.push_back(combination);
                // insert single items candidates
                for(int i=0; i<items.size(); i++) {
                    if(!(find(single_candidates.begin(), single_candidates.end(), items[i]) !=
single_candidates.end())){
                        single_candidates.push_back(items[i]);
                    }
                }
            }
            else{
cout << "Process " << rank << "---found combination--- " << combination << endl;
                break;
            }
        }
    }
}
```

<u>SERIAL</u>

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <sstream>
#include <map>
```

```
#include <algorithm>
#include <sys/time.h>
using namespace std;

void read_file(char file_name[], vector< vector<string> > &matrix, map<string,float>
&dictionary);
void find_itemsets(vector<string> matrix, vector<string> candidates, map<string,float>
&temp_dictionary, int k, int item_idx, string itemset, int current, vector<string>
single_candidates);
void prune_itemsets(map<string,float> &temp_dictionary, vector<string> &candidates, float
min_support, vector<string> &single_candidates);
void update_candidates(vector<string> &candidates, vector<string> freq_itemsets,
vector<string> &single_candidates);

int main(){
    char* file_name = "project.csv";
    float min_support = .05;
    vector< vector<string> > matrix;
    map<string,float> dictionary;
    map<string,float> temp_dictionary;
    vector<string> candidates;
    vector<string> single_candidates;
    int n_rows;
    string item;

    struct timeval start, end;
    double elapsed;

    gettimeofday(&start, NULL);

    // read file into 2D vector matrix and insert 1-itemsets in dictionary as key with their frequency
as value
    read_file(file_name, matrix, dictionary);
cout << "Read" << endl;
    n_rows = matrix.size();

    // divide frequency by number of rows to calculate support
    for (map<string, float>::iterator i = dictionary.begin(); i != dictionary.end(); ++i) {
        i->second = i->second/float(n_rows);
    }
```

```
    // prune from dictionary 1-itemsets with support < min_support and insert items in candidates
vector
    prune_itemsets(dictionary, candidates, min_support, single_candidates);

    // insert in dictionary all k-itemset
    int n = 2; // starting from 2-itemset
    while(!candidates.empty()){
        temp_dictionary.clear();
        // read matrix and insert n-itemsets in temp_dictionary as key with their frequency as value
        for (int i = 0; i < matrix.size(); i++){
            find_itemsets(matrix[i], candidates, temp_dictionary, n, -1, "", 0, single_candidates);
        }
        // divide frequency by number of rows to calculate support
        for (map<string, float>::iterator i = temp_dictionary.begin(); i != temp_dictionary.end();
++i) {
            i->second = i->second/float(n_rows);
        }
        // prune from temp_dictionary n-itemsets with support < min_support and insert items in
candidates vector
        prune_itemsets(temp_dictionary, candidates, min_support, single_candidates);
        // append new n-itemsets to main dictionary
        dictionary.insert(temp_dictionary.begin(), temp_dictionary.end());
        n++;
    }

    gettimeofday(&end, NULL);
    elapsed = (end.tv_sec - start.tv_sec) +
              ((end.tv_usec - start.tv_usec)/1000000.0);
    cout<<"Time passed: "<<elapsed<<endl;

    //removing extra space at the beginning of the word
    for (auto it = dictionary.begin(); it != dictionary.end(); ) {
        string& key = const_cast<string&>(it->first);
        size_t pos = key.find_first_not_of(' ');
        if (pos != string::npos) {
            key = key.substr(pos);
            it++;
        } else {
            it = dictionary.erase(it);
```

```
        }
    }

    // to sort the items on the basis of support value
    multimap<float,string> sorted_map;
    for (auto& it : dictionary){
            sorted_map.insert({it.second, it.first});
}

cout << endl << endl;
cout << "FREQUENT ITEMS ARE\n";

//printing the most frequent items where n=2
for (const auto& entry : sorted_map) {
    const float& value = entry.first;
    const string& key = entry.second;

    if (key.find(" ") != string::npos) {
        cout << key << endl;
    }
}

    return 0;
}

void read_file(char file_name[], vector< vector<string> > &matrix, map<string,float>
&dictionary){
    ifstream myfile (file_name);

    vector<string> row;
    string line;
    stringstream ss;
    string item;

    while(getline (myfile, line)){
        ss << line;

        while(getline (ss, item, ',')) {
            item.erase(remove(item.begin(), item.end(), '\r'), item.end());
            row.push_back(item);
```

```
            // insert item into dictionary and increment its value
            dictionary[item]++;
        }

        sort(row.begin(), row.end());
        matrix.push_back(row);

        ss.clear();
        row.clear();
    }

    myfile.close();
}

void find_itemsets(vector<string> matrix, vector<string> candidates, map<string,float>
&temp_dictionary, int k, int item_idx, string itemset, int current, vector<string>
single_candidates){
    if(current == k){
        itemset = itemset.erase(0,1); // remove first space

        // if itemset is a candidate insert it into temp_dictionary to calculate support
        if(find(candidates.begin(), candidates.end(), itemset) != candidates.end()){
            temp_dictionary[itemset]++;
            return;
        }
        // if itemset is not a candidate discard it
        else{
            return;
        }
    }

    string item;
    for (int j = ++item_idx; j < matrix.size(); j++){
        item = matrix[j];

        // if item does not compose one of the candidates, skip it
        if(!(find(single_candidates.begin(), single_candidates.end(), item) !=
single_candidates.end())){
            continue;
        }
```

```
        find_itemsets(matrix, candidates, temp_dictionary, k, j, itemset + " " + item, current+1,
single_candidates);
    }
}

void prune_itemsets(map<string,float> &temp_dictionary, vector<string> &candidates, float
min_support, vector<string> &single_candidates){
    vector<string> freq_itemsets;
    candidates.clear(); // empty candidates to then update it
    single_candidates.clear();

    for (map<string, float>::iterator it = temp_dictionary.begin(); it != temp_dictionary.end(); ){ //
like a while
        if (it->second < min_support){
            temp_dictionary.erase(it++);
        }
        else{
            freq_itemsets.push_back(it->first);
            ++it;
        }
    }

    if(!freq_itemsets.empty()){
        update_candidates(candidates, freq_itemsets, single_candidates);
    }
}

void update_candidates(vector<string> &candidates, vector<string> freq_itemsets,
vector<string> &single_candidates){
    string item;
    stringstream to_combine;
    vector<string> items;
    vector<string> elements;
    string combination;

    int common_items;

    for(int i = 0; i < freq_itemsets.size()-1; i++){
        for(int j = i+1; j < freq_itemsets.size(); j++){
```

```
            common_items = 0;
            items.clear();
            to_combine.clear();
            combination.clear();

            to_combine << freq_itemsets[i] + ',' + freq_itemsets[j];
            while(getline (to_combine, item, ',')) {
                if(!(find(items.begin(), items.end(), item) != items.end())){
                    items.push_back(item);
                    combination += " " + item;
                }
                else{
                    common_items++;
                }
            }

            // if the statement is true than we can add combination as candidate
            // else we created all correct combinations and we pass to the next itemset
            if(common_items == items.size()-2){
                combination.erase(0,1); // remove first space
                candidates.push_back(combination);

                // insert single items candidates
                for(int i=0; i<items.size(); i++) {
                    if(!(find(single_candidates.begin(), single_candidates.end(), items[i]) !=
single_candidates.end())){
                        single_candidates.push_back(items[i]);
                    }
                }
            }
            else{
                break;
            }

        }
    }
}
```

CONCLUSION

Parallelizing the Apriori algorithm can significantly improve its scalability and efficiency. The parallelized Apriori algorithm can be used to process large datasets in a reasonable amount of time.