**Apriori Algorithm Implementation**

using System.Text;

namespace AprioriAlgorithm

{

public class Apriori

{

private List<string> distinctValues;

private readonly List<string> itemlist;

private readonly List<ItemSet> itemSets;

public Apriori(string filePath)

{

itemlist = File.ReadAllLines(filePath).Where(a => !string.IsNullOrWhiteSpace(a)).ToList();

itemSets = new List<ItemSet>();

SetDistinctValues(itemlist);

}

public ItemSet GetItemSet(int length, int support, bool candidates = false, bool isFirstItemList = false)

{

List<IEnumerable<string>> result = GetPermutations(distinctValues, length).ToList();

List<List<string>> data = new List<List<string>>();

foreach (var item in result)

{

data.Add(item.ToList());

}

ItemSet itemSet = new ItemSet

{

Support = support,

Label = (candidates ? "C" : "L") + length.ToString()

};

foreach (var item in data)

{

int count = 0;

foreach (var word in itemlist)

{

bool found = false;

foreach (var item2 in item)

{

if (word.Split(' ').Contains(item2))

found = true;

else

{

found = false;

break;

}

}

if (found)

count++;

}

if ((candidates && count > 0) || isFirstItemList || count >= support)

{

itemSet.Add(item, count);

itemSets.Add(itemSet);

}

}

return itemSet;

}

public List<AssociationRule> GetAssociationRules(ItemSet itemSet)

{

var associationRules = new List<AssociationRule>();

foreach (var item in itemSet)

{

foreach (var set in item.Key)

{

associationRules.Add(GetAssociationRule(set, item));

if (item.Key.Count > 2)

{

associationRules.Add(GetAssociationRule(item.Key.ToDisplay(exclude: set), item));

}

}

}

return associationRules

.OrderByDescending(a => a.Support)

.ThenByDescending(a => a.Confidance)

.ToList();

}

private AssociationRule GetAssociationRule(string set, KeyValuePair<List<string>, int> item)

{

var setItems = set.Split(',');

for (int i = 0; i < setItems.Count(); i++)

{

setItems[i] = setItems[i].Trim();

}

var associationRule = new AssociationRule();

var sb = new StringBuilder();

sb.Append(set).Append(" => ");

var list = new List<string>();

foreach (var set2 in item.Key)

{

if (setItems.Contains(set2)) continue;

list.Add(set2);

}

sb.Append(list.ToDisplay());

associationRule.Label = sb.ToString();

int totalSet = 0;

foreach (var first in itemSets)

{

var myItem = first.Keys.Where(a => a.ToDisplay() == set);

if (myItem.Any())

{

first.TryGetValue(myItem.FirstOrDefault(), out totalSet);

break;

}

}

associationRule.Confidance = Math.Round(((double)item.Value / totalSet) \* 100, 2);

associationRule.Support = Math.Round(((double)item.Value / this.itemlist.Count) \* 100, 2);

return associationRule;

}

private void SetDistinctValues(List<string> values)

{

var data = new List<string>();

foreach (var item in values)

{

var row = item.Split(' ');

foreach (var item2 in row)

{

if (string.IsNullOrWhiteSpace(item2)) continue;

if (!data.Contains(item2))

data.Add(item2);

}

}

distinctValues = new List<string>();

distinctValues.AddRange(data.OrderBy(a => a).ToList());

}

private static IEnumerable<IEnumerable<T>> GetPermutations<T>(IEnumerable<T> items, int count)

{

int i = 0;

foreach (var item in items)

{

if (count == 1)

{

yield return new T[] { item };

}

else

{

foreach (var result in GetPermutations(items.Skip(i + 1), count - 1))

yield return new T[] { item }.Concat(result);

}

++i;

}

}

}

public class AssociationRule

{

public string Label { get; set; }

public double Confidance { get; set; }

public double Support { get; set; }

}

public class ItemSet : Dictionary<List<string>, int>

{

public string Label { get; set; }

public int Support { get; set; }

}

public class TableUserControl

{

public TableUserControl(ItemSet itemSet, List<AssociationRule> rules)

{

if (itemSet.Any())

{

Console.WriteLine(itemSet.Label);

Console.WriteLine($"Item Set\t\tCount");

Console.WriteLine("------------------------------------------------------");

foreach (var item in itemSet)

{

Console.WriteLine($"{item.Key.ToDisplay()}\t\t{item.Value}");

}

Console.WriteLine("\n");

Console.WriteLine("------------------------------------------------------");

Console.WriteLine("\n");

}

if (rules.Any())

{

Console.WriteLine($"Rule\t\tConfidence\t\tSupport");

Console.WriteLine("------------------------------------------------------");

foreach (var item in rules)

{

Console.WriteLine($"{item.Label}\t\t{item.Confidance.ToPercentString()}\t\t{item.Support.ToPercentString()}");

}

Console.WriteLine("\n");

Console.WriteLine("------------------------------------------------------");

Console.WriteLine("\n");

}

}

public TableUserControl(List<string> values)

{

if (values.Any())

{

Console.WriteLine("Transactions");

Console.WriteLine($"Item Set\t\tCount");

Console.WriteLine("------------------------------------------------------");

for (int i = 0; i < values.Count; i++)

{

Console.WriteLine($"{i}\t\t{values[i]}");

}

Console.WriteLine("\n");

Console.WriteLine("------------------------------------------------------");

Console.WriteLine("\n");

}

}

}

public class Program

{

static string FileName = "demo.txt";

static void Main()

{

DoThings();

Console.ReadKey();

}

static void DoThings()

{

int support = 2;

var filePath = Path.Combine(Environment.CurrentDirectory, FileName);

new TableUserControl(File.ReadAllLines(filePath).ToList());

var apriori = new Apriori(FileName);

int k = 1;

var itemSets = new List<ItemSet>();

bool next;

do

{

next = false;

var itemSet = apriori.GetItemSet(k, support, isFirstItemList: k == 1);

if (itemSet.Count > 0)

{

var associationRules = new List<AssociationRule>();

if (k != 1)

{

associationRules = apriori.GetAssociationRules(itemSet);

}

new TableUserControl(itemSet, associationRules);

next = true;

k++;

itemSets.Add(itemSet);

}

} while (next);

}

}

public static class Helper

{

public static string ToDisplay(this List<string> list, string separator = ", ")

{

if (list.Count == 0)

return string.Empty;

StringBuilder sb = new StringBuilder();

sb.Append(list[0]);

for (int i = 1; i < list.Count; i++)

{

sb.Append(string.Format("{0}{1}", separator, list[i]));

}

return sb.ToString();

}

public static string ToDisplay(this List<string> list, string exclude, string separator = ", ")

{

List<string> dump = new List<string>();

foreach (var item in list)

{

if (item == exclude) continue;

dump.Add(item.ToString());

}

return dump.ToDisplay();

}

public static string ToPercentString(this object item)

{

return item.ToString() + " %";

}

}

}

Input:

A F

A B F

A C F

A B C F

S F D A S

B C S

B D A

B F S

D S F

A V C D R

Output:

Transactions

Item Set Count

------------------------------------------------------

0 A F

1 A B F

2 A C F

3 A B C F

4 S F D A S

5 B C S

6 B D A

7 B F S

8 D S F

9 A V C D R

------------------------------------------------------

L1

Item Set Count

------------------------------------------------------

A 7

B 5

C 4

D 4

F 7

R 1

S 4

V 1

------------------------------------------------------

L2

Item Set Count

------------------------------------------------------

A, B 3

A, C 3

A, D 3

A, F 5

B, C 2

B, F 3

B, S 2

C, F 2

D, F 2

D, S 2

F, S 3

------------------------------------------------------

Rule Confidence Support

------------------------------------------------------

A => F 71.43 % 50 %

F => A 71.43 % 50 %

C => A 75 % 30 %

D => A 75 % 30 %

S => F 75 % 30 %

B => A 60 % 30 %

B => F 60 % 30 %

A => B 42.86 % 30 %

A => C 42.86 % 30 %

A => D 42.86 % 30 %

F => B 42.86 % 30 %

F => S 42.86 % 30 %

C => B 50 % 20 %

S => B 50 % 20 %

C => F 50 % 20 %

D => F 50 % 20 %

D => S 50 % 20 %

S => D 50 % 20 %

B => C 40 % 20 %

B => S 40 % 20 %

F => C 28.57 % 20 %

F => D 28.57 % 20 %

------------------------------------------------------

L3

Item Set Count

------------------------------------------------------

A, B, F 2

A, C, F 2

D, F, S 2

------------------------------------------------------

Rule Confidence Support

------------------------------------------------------

C, F => A 100 % 20 %

D, S => F 100 % 20 %

D, F => S 100 % 20 %

B, F => A 66.67 % 20 %

A, B => F 66.67 % 20 %

A, C => F 66.67 % 20 %

F, S => D 66.67 % 20 %

C => A, F 50 % 20 %

D => F, S 50 % 20 %

S => D, F 50 % 20 %

B => A, F 40 % 20 %

A, F => B 40 % 20 %

A, F => C 40 % 20 %

A => B, F 28.57 % 20 %

F => A, B 28.57 % 20 %

A => C, F 28.57 % 20 %

F => A, C 28.57 % 20 %

F => D, S 28.57 % 20 %

------------------------------------------------------