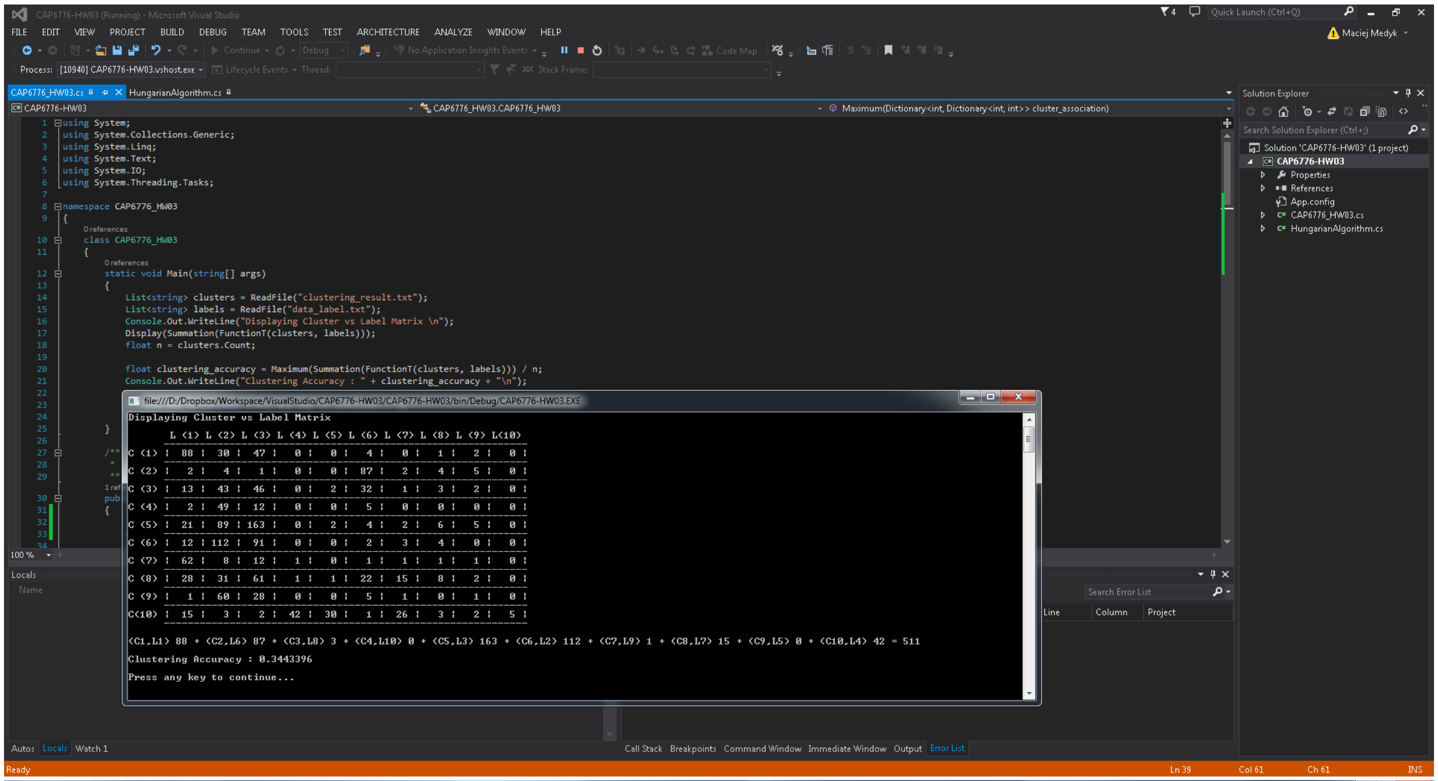
Maciej Medyk – CAP6776 – Information Retrieval – Homework 03

Report – [10.00pt] – Calculate Clustering Accuracy for Provided Data Sets

Clustering Accuracy running result is 0.3443396 or 34.434% in both C Sharp and Java applications
I implemented C Sharp program using Hungarian Algorithm while in Java program I used greedy algorithm.
Hungarian is more reliable algorithm as it results in optimal solution every time while greedy is less reliable.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Threading.Tasks;

namespace CAP6776_HW03
{
    class CAP6776_HW03
    {
        static void Main(string[] args)
        {
            List<string> clusters = ReadFile("clustering_result.txt");
            List<string> labels = ReadFile("data_label.txt");
            Console.Out.WriteLine("Displaying Cluster vs Label Matrix \n");
            Display(Summation(FunctionT(clusters, labels)));
            float n = clusters.Count;

            float clustering_accuracy = Maximum(Summation(FunctionT(clusters, labels))) / n;
            Console.Out.WriteLine("Clustering Accuracy : " + clustering_accuracy + "\n");
```

file:///D:/Dropbox/Workspace/VisualStudio/CAP6776-HW03/CAP6776-HW03/bin/Debug/CAP6776-HW03.EXE

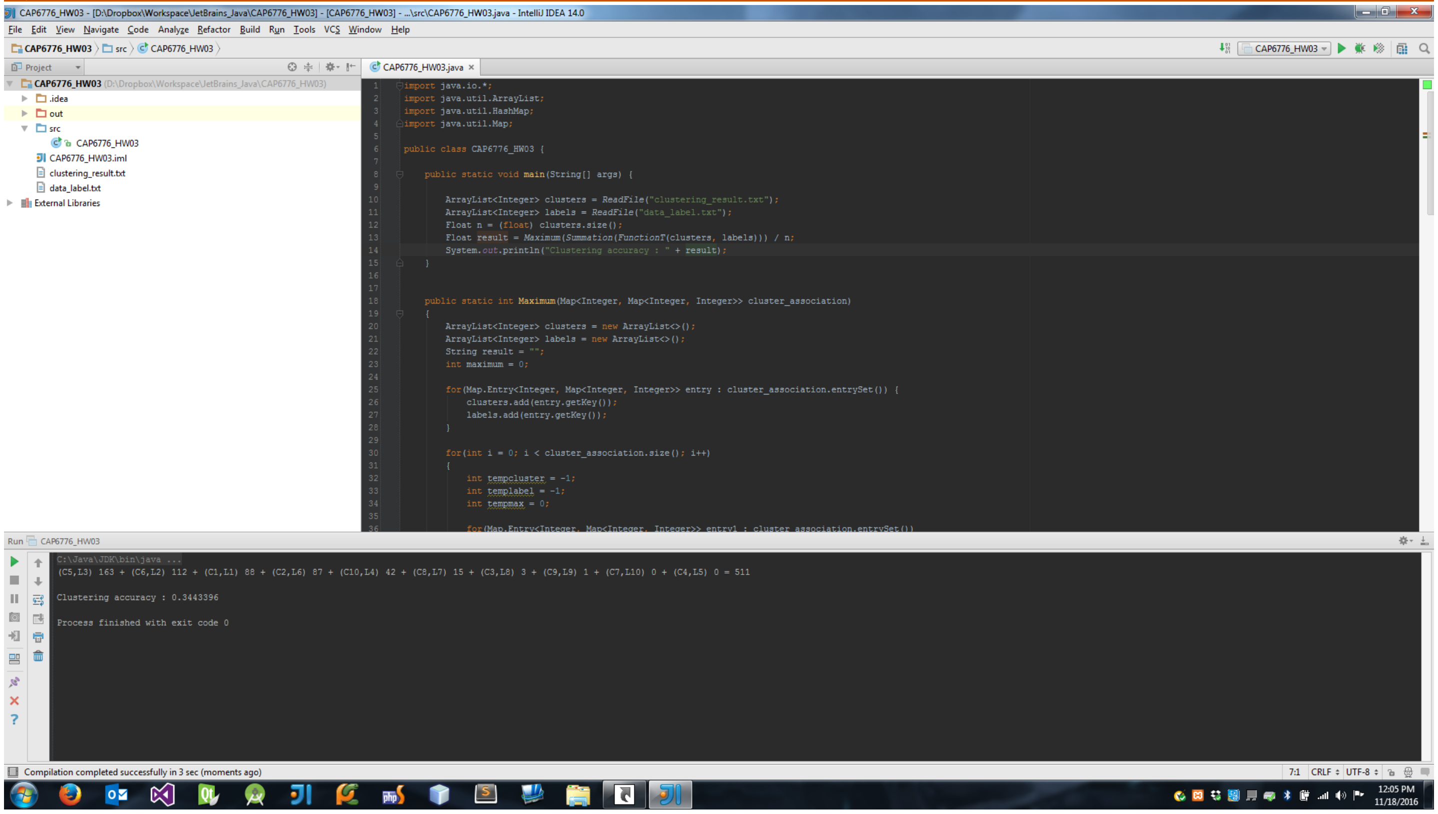
```
Displaying Cluster vs Label Matrix

      L (1) L (2) L (3) L (4) L (5) L (6) L (7) L (8) L (9) L(10)
     -----------------------------------------------------------
C (1) ¦  88 ¦  30 ¦  47 ¦   0 ¦   0 ¦   4 ¦   0 ¦   1 ¦   2 ¦   0 ¦
     -----------------------------------------------------------
C (2) ¦   2 ¦   4 ¦   1 ¦   0 ¦   0 ¦  87 ¦   2 ¦   4 ¦   5 ¦   0 ¦
     -----------------------------------------------------------
C (3) ¦  13 ¦  43 ¦  46 ¦   0 ¦   2 ¦  32 ¦   1 ¦   3 ¦   2 ¦   0 ¦
     -----------------------------------------------------------
C (4) ¦   2 ¦  49 ¦  12 ¦   0 ¦   0 ¦   5 ¦   0 ¦   0 ¦   0 ¦   0 ¦
     -----------------------------------------------------------
C (5) ¦  21 ¦  89 ¦ 163 ¦   0 ¦   2 ¦   4 ¦   2 ¦   6 ¦   5 ¦   0 ¦
     -----------------------------------------------------------
C (6) ¦  12 ¦ 112 ¦  91 ¦   0 ¦   0 ¦   2 ¦   3 ¦   4 ¦   0 ¦   0 ¦
     -----------------------------------------------------------
C (7) ¦  62 ¦   8 ¦  12 ¦   1 ¦   0 ¦   1 ¦   1 ¦   1 ¦   1 ¦   0 ¦
     -----------------------------------------------------------
C (8) ¦  28 ¦  31 ¦  61 ¦   1 ¦   1 ¦  22 ¦  15 ¦   8 ¦   2 ¦   0 ¦
     -----------------------------------------------------------
C (9) ¦   1 ¦  60 ¦  28 ¦   0 ¦   0 ¦   5 ¦   1 ¦   0 ¦   1 ¦   0 ¦
     -----------------------------------------------------------
C(10) ¦  15 ¦   3 ¦   2 ¦  42 ¦  30 ¦   1 ¦  26 ¦   3 ¦   2 ¦   5 ¦
     -----------------------------------------------------------

(C1,L1) 88 + (C2,L6) 87 + (C3,L8) 3 + (C4,L10) 0 + (C5,L3) 163 + (C6,L2) 112 + (C7,L9) 1 + (C8,L7) 15 + (C9,L5) 0 + (C10,L4) 42 = 511

Clustering Accuracy : 0.3443396

Press any key to continue...
```



```
import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

public class CAP6776_HW03 {

    public static void main(String[] args) {

        ArrayList<Integer> clusters = ReadFile("clustering_result.txt");
        ArrayList<Integer> labels = ReadFile("data_label.txt");
        Float n = (float) clusters.size();
        Float result = Maximum(Summation(FunctionT(clusters, labels))) / n;
        System.out.println("Clustering accuracy : " + result);
    }


    public static int Maximum(Map<Integer, Map<Integer, Integer>> cluster_association)
    {
        ArrayList<Integer> clusters = new ArrayList<>();
        ArrayList<Integer> labels = new ArrayList<>();
        String result = "";
        int maximum = 0;

        for(Map.Entry<Integer, Map<Integer, Integer>> entry : cluster_association.entrySet()) {
            clusters.add(entry.getKey());
            labels.add(entry.getKey());
        }

        for(int i = 0; i < cluster_association.size(); i++)
        {
            int tempcluster = -1;
            int templabel = -1;
            int tempmax = 0;

            for(Map.Entry<Integer, Map<Integer, Integer>> entry1 : cluster_association.entrySet())
```

```
C:\Java\JDK\bin\java ...
(C5,L3) 163 + (C6,L2) 112 + (C1,L1) 88 + (C2,L6) 87 + (C10,L4) 42 + (C8,L7) 15 + (C3,L8) 3 + (C9,L9) 1 + (C7,L10) 0 + (C4,L5) 0 = 511

Clustering accuracy : 0.3443396

Process finished with exit code 0
```

Addendum – Program Code Using C# with Hungarian Algorithm - Program Zipped Code Included on BlackBoard

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Threading.Tasks;

namespace CAP6776_HW03
{
    class CAP6776_HW03
    {
        static void Main(string[] args)
        {
            List<string> clusters = ReadFile("clustering_result.txt");
            List<string> labels = ReadFile("data_label.txt");
            Console.Out.WriteLine("Displaying Cluster vs Label Matrix \n");
            Display(Summation(FunctionT(clusters, labels)));
            float n = clusters.Count;

            float clustering_accuracy = Maximum(Summation(FunctionT(clusters, labels))) / n;
            Console.Out.WriteLine("Clustering Accuracy : " + clustering_accuracy + "\n");

            Console.Out.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

        /**
         *  Extracts maximum counts from each cluster and returns maximum integer
         **/
        public static int Maximum(Dictionary<int, Dictionary<int, int>> cluster_association)
        {

            string result = "";
            int maximum = 0;

            int size = cluster_association.Count;
            int maxvalue = 0;
            int[,] array = new int[size, size];
            for (int i = 1; i <= cluster_association.Count; i++)
            {
                for (int j = 1; j <= cluster_association.Count; j++)
                {
                    int tempInt;
                    Dictionary<int, int> tempDict;
                    cluster_association.TryGetValue(i, out tempDict);
                    tempDict.TryGetValue(j, out tempInt);
                    if (tempInt > maxvalue)
                    {
                        maxvalue = tempInt;
                    }
                }
            }

            for (int i = 1; i <= cluster_association.Count; i++)
            {
                for (int j = 1; j <= cluster_association.Count; j++)
                {
                    int tempInt;
                    Dictionary<int, int> tempDict;
                    cluster_association.TryGetValue(i, out tempDict);
                    tempDict.TryGetValue(j, out tempInt);
                    array[i - 1, j - 1] = maxvalue - tempInt;
                }
            }

            HungarianAlgorithm hungarian = new HungarianAlgorithm(array);
            int[] hungarianresults = hungarian.Evaluate();

            for (int i = 0; i < hungarianresults.Length; i++)
            {
                maximum += cluster_association[i + 1][hungarianresults[i] + 1];
                result += "(C" + (i + 1) + ",L" + (hungarianresults[i] + 1) + ") " + cluster_association[i + 1][hungarianresults[i] +
1] + " + ";
            }

            result = result.Substring(0, result.Length - 3) + " = " + maximum + "\n";
            Console.Out.WriteLine(result);
            return maximum;
        }

        /**
         *  Summarizes counts for all classes within clusters and returns dictionary of cluster counts
         **/
        public static Dictionary<int, Dictionary<int, int>> Summation(Dictionary<string, List<string>> cluster_association)
        {
            Dictionary<int, Dictionary<int, int>> result2D = new Dictionary<int, Dictionary<int, int>>();

            foreach (KeyValuePair<string, List<string>> entry in cluster_association)
            {
                int parsedKey = int.Parse(entry.Key);
                Dictionary<int, int> result1D = new Dictionary<int, int>();

                foreach (KeyValuePair<string, List<string>> subentry in cluster_association)
                {
                    int parsedKeyValue = int.Parse(subentry.Key);
                    result1D.Add(parsedKeyValue, 0);
                }

                for (int i = 0; i < entry.Value.Count; i++)
                {
                    int parsedKeyValue = int.Parse(entry.Value[i]);
                    result1D[parsedKeyValue] += 1;
                }
                result2D.Add(parsedKey, result1D);
            }
            return result2D;
        }

        /**
         *  Associates all clusters with classes and returns list of clusters
         **/
        public static Dictionary<string, List<string>> FunctionT(List<string> clusters, List<string> labels)
        {
            Dictionary<string, List<string>> cluster_association = new Dictionary<string, List<string>>();

            for (int i = 0; i < clusters.Count; i++)
            {
                if (cluster_association.ContainsKey(clusters[i]))
                {
                    List<string> temp = cluster_association[clusters[i]];
                    temp.Add(labels[i]);
                    cluster_association[clusters[i]] = temp;
                }
                else
                {
                    List<string> temp = new List<string>();
                    temp.Add(labels[i]);
                    cluster_association.Add(clusters[i], temp);
                }
            }
            return cluster_association;
        }

        /**
         *  Reads files from the program and returns lists of strings
         **/
        public static List<string> ReadFile(String filename)
        {
            List<string> result = new List<string>();
            var lines = File.ReadLines(filename);
            foreach (var line in lines)
            {
                result.Add(line.Trim());
            }
            return result;
        }

        /**
         *  Displays the matrix of cluster class count allocation
         **/
        public static void Display(Dictionary<int, Dictionary<int, int>> cluster_association)
        {
            Console.Out.Write(SpaceOut("") + "  ");
            for (int i = 1; i <= cluster_association.Count; i++)
            {
                Console.Out.Write("L" + SpaceOut("(" + i + ")", 4, 0) + " ");
            }
            Console.Out.Write("\n");
            Console.Out.WriteLine(SpaceOut("") + " -------------------------------------------------------------");

            for(int i = 1; i <= cluster_association.Count; i++)
            {
                Console.Out.Write("C" + SpaceOut("(" + i  + ")"));
                Console.Out.Write("|");
                for(int j = 1; j <= cluster_association.Count; j++)
                {
                    int tempInt;
                    Dictionary<int, int> tempDict;
                    cluster_association.TryGetValue(i, out tempDict);
                    tempDict.TryGetValue(j, out tempInt);
                    Console.Out.Write(SpaceOut(tempInt.ToString()) + "|");
                }
                Console.Out.Write("\n");
                Console.Out.WriteLine(SpaceOut("") + " -------------------------------------------------------------");
            }
            Console.Out.Write("\n");
        }

        /**
         *  Spacing out for strings
         */
        private static string SpaceOut(string input, int total = 5, int rmargin = 1)
        {
            string result = "";
            string margin = "";
            for (int i = 0; i < rmargin; i++) margin += " ";
            input = input + margin;
            int size = input.Length;
            for (int i = 0; i < total - size; i++) result += " ";
            return result + input;
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CAP6776_HW03
{
    class HungarianAlgorithm
    {
        private readonly int[,] matrix;
        private int maxvalue;
        private int numberofelements;
        private int[] clusters; //labels for workers
        private int[] labels; //labels for jobs
        private bool[] clustervertices;
        private bool[] labelvertices;
        private int[] clusterscovered; //vertex matched with x
        private int[] labelscovered; //vertex matched with y
        private int totalcovered;
        private int[] yslack;
        private int[] xslack;
        private int[] paths; //memorizing paths

        public HungarianAlgorithm(int[,] matrixIn)
        {
            matrix = matrixIn;
        }

        public int[] Evaluate()
        {
            numberofelements = matrix.GetLength(0);

            clusters = new int[numberofelements];
            labels = new int[numberofelements];
            clustervertices = new bool[numberofelements];
            labelvertices = new bool[numberofelements];
            clusterscovered = new int[numberofelements];
            labelscovered = new int[numberofelements];
            yslack = new int[numberofelements];
            xslack = new int[numberofelements];
            paths = new int[numberofelements];
            maxvalue = int.MaxValue;


            InittiateMatches();

            if (numberofelements != matrix.GetLength(1))
                return null;

            IntitiateLabels();

            totalcovered = 0;

            InitialMatching();

            var queue = new Queue<int>();

            #region augment

            while (totalcovered != numberofelements)
            {
                queue.Clear();

                InitiateVertices();

                var root = 0;
                int xaxis;
                var yaxis = 0;

                for (xaxis = 0; xaxis < numberofelements; xaxis++)
                {
                    if (clusterscovered[xaxis] != -1) continue;
                    queue.Enqueue(xaxis);
                    root = xaxis;
                    paths[xaxis] = -2;

                    clustervertices[xaxis] = true;
                    break;
                }
                for (var i = 0; i < numberofelements; i++)
                {
                    yslack[i] = matrix[root, i] - clusters[root] - labels[i];
                    xslack[i] = root;
                }

                while (true)
                {
                    while (queue.Count != 0)
                    {
                        xaxis = queue.Dequeue();
                        var lxx = clusters[xaxis];
                        for (yaxis = 0; yaxis < numberofelements; yaxis++)
                        {
                            if (matrix[xaxis, yaxis] != lxx + labels[yaxis] || labelvertices[yaxis]) continue;
                            if (labelscovered[yaxis] == -1) break;
                            labelvertices[yaxis] = true;
                            queue.Enqueue(labelscovered[yaxis]);

                            AddToTree(labelscovered[yaxis], xaxis);
                        }
                        if (yaxis < numberofelements) break;
                    }
                    if (yaxis < numberofelements) break;

                    UpdateLabels();

                    for (yaxis = 0; yaxis < numberofelements; yaxis++)
                    {
                        if (labelvertices[yaxis] || yslack[yaxis] != 0) continue;
                        if (labelscovered[yaxis] == -1)
                        {
                            xaxis = xslack[yaxis];
                            break;
                        }
                        labelvertices[yaxis] = true;
                        if (clustervertices[labelscovered[yaxis]]) continue;
                        queue.Enqueue(labelscovered[yaxis]);
                        AddToTree(labelscovered[yaxis], xslack[yaxis]);
                    }
                    if (yaxis < numberofelements) break;
                }

                totalcovered++;

                int totaly;
                for (int cx = xaxis, cy = yaxis; cx != -2; cx = paths[cx], cy = totaly)
                {
                    totaly = clusterscovered[cx];
                    labelscovered[cy] = cx;
                    clusterscovered[cx] = cy;
                }
            }

            #endregion

            return clusterscovered;
        }

        private void InittiateMatches()
        {
            for (var i = 0; i < numberofelements; i++)
            {
                clusterscovered[i] = -1;
                labelscovered[i] = -1;
            }
        }

        private void InitiateVertices()
        {
            for (var i = 0; i < numberofelements; i++)
            {
                clustervertices[i] = false;
                labelvertices[i] = false;
            }
        }

        private void IntitiateLabels()
        {
            for (var i = 0; i < numberofelements; i++)
            {
                var minRow = matrix[i, 0];
                for (var j = 0; j < numberofelements; j++)
                {
                    if (matrix[i, j] < minRow) minRow = matrix[i, j];
                    if (minRow == 0) break;
                }
                clusters[i] = minRow;
            }
            for (var j = 0; j < numberofelements; j++)
            {
                var minColumn = matrix[0, j] - clusters[0];
                for (var i = 0; i < numberofelements; i++)
                {
                    if (matrix[i, j] - clusters[i] < minColumn) minColumn = matrix[i, j] - clusters[i];
                    if (minColumn == 0) break;
                }
                labels[j] = minColumn;
            }
        }

        private void UpdateLabels()
        {
            var delta = maxvalue;
            for (var i = 0; i < numberofelements; i++)
                if (!labelvertices[i])
                    if (delta > yslack[i])
                        delta = yslack[i];
            for (var i = 0; i < numberofelements; i++)
            {
                if (clustervertices[i])
                    clusters[i] = clusters[i] + delta;
                if (labelvertices[i])
                    labels[i] = labels[i] - delta;
                else yslack[i] = yslack[i] - delta;
            }
        }

        private void AddToTree(int x, int prevx)
        {

            clustervertices[x] = true;
            paths[x] = prevx;

            var lxx = clusters[x];
            for (var y = 0; y < numberofelements; y++)
            {
                if (matrix[x, y] - lxx - labels[y] >= yslack[y]) continue;
                yslack[y] = matrix[x, y] - lxx - labels[y];
                xslack[y] = x;
            }
        }

        private void InitialMatching()
        {
            for (var x = 0; x < numberofelements; x++)
            {
                for (var y = 0; y < numberofelements; y++)
                {
                    if (matrix[x, y] != clusters[x] + labels[y] || labelscovered[y] != -1) continue;
                    clusterscovered[x] = y;
                    labelscovered[y] = x;
                    totalcovered++;
                    break;
                }
            }
        }
    }
}
```

Addendum – Program Code Using Java using Greedy Algorithm - Program Zipped Code Included on BlackBoard

```
import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

public class CAP6776_HW03 {

    public static void main(String[] args) {

        ArrayList<Integer> clusters = ReadFile("clustering_result.txt");
        ArrayList<Integer> labels = ReadFile("data_label.txt");
        Float n = (float) clusters.size();
        Float result = Maximum(Summation(FunctionT(clusters, labels))) / n;
        System.out.println("Clustering accuracy : " + result);
    }


    public static int Maximum(Map<Integer, Map<Integer, Integer>> cluster_association)
    {
        ArrayList<Integer> clusters = new ArrayList<>();
        ArrayList<Integer> labels = new ArrayList<>();
        String result = "";
        int maximum = 0;

        for(Map.Entry<Integer, Map<Integer, Integer>> entry : cluster_association.entrySet()) {
            clusters.add(entry.getKey());
            labels.add(entry.getKey());
        }

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

```java
            int templabel = -1;
            int tempmax = 0;

            for(Map.Entry<Integer, Map<Integer, Integer>> entry1 : cluster_association.entrySet())
            {
                for(Map.Entry<Integer, Integer> entry2 : entry1.getValue().entrySet())
                {
                    if(entry2.getValue() >= tempmax && clusters.contains(entry1.getKey()) && labels.contains(entry2.getKey()))
                    {
                        tempcluster = entry1.getKey();
                        templabel = entry2.getKey();
                        tempmax = entry2.getValue();
                    }
                }
            }
            clusters.remove(clusters.indexOf(tempcluster));
            labels.remove(labels.indexOf(templabel));
            result += "(C" + tempcluster + ",L" + templabel + ") " + tempmax + " + ";
            maximum += tempmax;
        }
        result = result.substring(0, result.length() - 3) + " = " + maximum + "\n";
        System.out.println(result);
        return maximum;
    }


    public static Map<Integer, Map<Integer, Integer>> Summation(Map<Integer, ArrayList<Integer>> cluster_association)
    {
        Map<Integer, Map<Integer, Integer>> result = new HashMap<>();

        for(Map.Entry<Integer, ArrayList<Integer>> entry1 : cluster_association.entrySet()) {

            Integer parsedKey = entry1.getKey();
            Map<Integer, Integer> tempresult = new HashMap<>();

            for(Map.Entry<Integer, ArrayList<Integer>> entry2 : cluster_association.entrySet()) {
                Integer parsedKeyValue = entry2.getKey();
                tempresult.put(parsedKeyValue, 0);
            }

            for(int i = 0; i < entry1.getValue().size(); i++)
            {
                int parsedKeyValue = entry1.getValue().get(i);
                Integer value = tempresult.get(parsedKeyValue) + 1;
                tempresult.put(parsedKeyValue, value);
            }
            result.put(parsedKey, tempresult);
        }

        return result;
    }


    public static Map<Integer, ArrayList<Integer>> FunctionT(ArrayList<Integer> clusters, ArrayList<Integer> labels) {

        Map<Integer, ArrayList<Integer>> result = new HashMap<>();
        for(int i = 0; i < clusters.size(); i++) {

            if (result.containsKey(clusters.get(i)))
            {
                ArrayList<Integer> temp = result.get(clusters.get(i));
                temp.add(labels.get(i));
                result.put(clusters.get(i), temp);
            }
            else {
                ArrayList<Integer> temp = new ArrayList<>();
                temp.add(labels.get(i));
                result.put(clusters.get(i), temp);
            }
        }
        return result;
    }

    public static ArrayList<Integer> ReadFile(String filename) {

        ArrayList<Integer> result = new ArrayList<>();

        try(BufferedReader br = new BufferedReader(new FileReader(filename))) {
            for(String line; (line = br.readLine()) != null; ) {
                result.add(Integer.parseInt(line.trim()));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

        return result;
    }
}
```