Appendix A

Simi_degreeApp

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Xml;
using System.IO;
using System.Linq;
using System.Collections;
using System.Text.RegularExpressions;
using System.Data;
using System.Data.SqlClient;

namespace CompareXML
{

    class Program
    {

        static void Main(string[] args)
        {
            string[] name_from_base;
            string[] name_from_external;
            string path = @"C:\MyTest.txt";
            string path2 = @"C:\MyTest2.txt";
            string path3 = @"C:\result\Myresult2.txt";

             float semi_degree = 0.0f;
            ArrayList arrText = new ArrayList();
            ArrayList list = new ArrayList();
            ArrayList list2 = new ArrayList();

            ArrayList name = new ArrayList();
            List<string> distinct_from_base = new List<string>();
            List<string> distinct_from_extern = new List<string>();


            XmlDataDocument doc = new XmlDataDocument();
            doc.Load("C:\\s5a.xml");
            XmlElement root = doc.DocumentElement;
            XmlNodeList elemList = root.GetElementsByTagName("RECORD");


            XmlTextReader reader2 = new XmlTextReader("c:\\d10.xml" );

            var nodes = doc.SelectNodes("S/RECORD");

            if (!File.Exists(path) && !File.Exists(path2))
            {
                // Create a file to write to.
```

```
                using (StreamWriter sw = File.CreateText(path))
                using (StreamWriter pw = File.CreateText(path2))
                {

                    XmlTextReader reader = new
XmlTextReader("c:\\d10.xml");
                    XmlTextReader reader1 = new
XmlTextReader("C:\\s5a.xml");

                    while (reader.Read())
                    {
                        XmlNodeType nodeType = reader.NodeType;
                        switch (nodeType)
                        {

                            case XmlNodeType.Element:
sw.WriteLine(reader.Name);
                                break;
                        }
                    }
                    while (reader1.Read())
                    {

                        XmlNodeType nodeType1 = reader1.NodeType;
                        switch (nodeType1)
                        {
                            case XmlNodeType.Element:
pw.WriteLine(reader1.Name);

                                break;

                        }

                    }

                }
            }



            Hashtable ht = new Hashtable();
            Hashtable ht1 = new Hashtable();

            string file1 = @"c:\d10.xml";
            string file2 = @"C:\s5a.xml";

            if (File.Exists(file1) && File.Exists(file2))
            {
                XmlTextReader rdrXml2 = new XmlTextReader(file2);
                XmlTextReader rdrXml1 = new XmlTextReader(file1);

                while (rdrXml2.Read())
                {
                    switch (rdrXml2.NodeType)
                    {
```

```
                        case XmlNodeType.Element:
                            list2.Add(rdrXml2.Name);
                            list2.Remove("d5");
                            list2.Remove("RECORD");
                            list2.Remove("D5");

                            break;

                    }
                }
                while (rdrXml1.Read())
                {
                    switch (rdrXml1.NodeType)
                    {
                        case XmlNodeType.Element:
                            list.Add(rdrXml1.Name);
                            list.Remove("d10 " );
                            list.Remove("RECORD");
                            list.Remove("D10" );

                            break;

                    }

                }



                foreach (string item2 in list2)
                {
                    ht[item2] = null;
                }
                ArrayList distinclist2 = new ArrayList(ht.Keys);
                list2.Clear();
                name_from_base = distinclist2.ToArray(typeof(string))
as string[];

                foreach (string item in list)
                {
                    ht1[item] = null;
                }
                ArrayList distincArray = new ArrayList(ht1.Keys);

                list.Clear();
                name_from_external =
distincArray.ToArray(typeof(string)) as string[];



                foreach (string ele in name_from_base)
                {


                    if (name_from_external.Contains(ele))
                    {
```

```
                    name.Add(ele);

                }

            }


            string[] same_item = name.ToArray(typeof(string)) as
string[];


            name.Clear();


            semi_degree = ((float)name_from_base.Count() +
(float)name_from_external.Count()) / (float)same_item.Count();


            Console.WriteLine("base " + name_from_base.Count());
            Console.WriteLine("external " +
name_from_external.Count());
            Console.WriteLine("inter {0} ", same_item.Count());


            using (StreamWriter rw = File.AppendText(path3))
            {


                rw.WriteLine(semi_degree);

                rw.Close();

            }

            /*}*/
            Dictionary<string, double> frequencyTable1 =
PrepareFrequency(name_from_base);
            Dictionary<string, double> frequencyTable2 =
PrepareFrequency(name_from_external);



            Dictionary<string, double> tfTable1 =
TfFactorized(frequencyTable1);
            Dictionary<string, double> tfTable2 =
TfFactorized(frequencyTable2);




            Dictionary<string, double>[] tables = new
Dictionary<string, double>[2];
            tables[0] = tfTable1;
            tables[1] = tfTable2;

            PrepareAllHashTables(tables);
```

```
                Console.WriteLine(CosineSimilarity(tfTable1,
tfTable2));

                tables =
GetPreparedTFIDFTables(IDFDocumentTable(tables), tables);


                Console.WriteLine(CosineSimilarity(tables[0],
tables[1]));


            }
        }

        private static string[] EatWhiteChar(string sentence)
        {
            char[] cSentence = sentence.ToCharArray();

            ArrayList wordList = new ArrayList();
            int index = 0;

            string word = "";
            while (index < sentence.Length)
            {

                while (index < sentence.Length && ((cSentence[index] ==
' ' || cSentence[index] == '\n' || cSentence[index] == '\r' ||
cSentence[index] == '\t')))
                    index++;

                while (index < cSentence.Length && ((cSentence[index]
!= ' ' && cSentence[index] != '\n' && cSentence[index] != '\r' &&
cSentence[index] != '\t')))
                    word += cSentence[index++];

                if (word != "")
                {
                    wordList.Add(word);
                    word = "";
                }


            }

            return (string[])wordList.ToArray(typeof(string));
        }
        private static  string FromSource(string source)
        {
            StreamReader rdr = new StreamReader(source);
            string text = rdr.ReadToEnd();

            rdr.Close();

            return text;
        }

        private static void WriteToFile(string[] words)
```

```
        {
            StreamWriter wrt = new StreamWriter("c:\\Test1.txt");
            foreach (string word in words)
            {
                wrt.WriteLine(word);
            }

            wrt.Close();
        }

        private static Dictionary<string, double>
PrepareFrequency(string[] words)
        {
            Dictionary<string,double> table = new
Dictionary<string,double>();

            foreach (string word in words)
            {
                if (table.ContainsKey(word))
                    table[word]++;
                else
                    table.Add(word, 1);
            }

            return table;
        }

       private static Dictionary<string,double>
TfFactorized(Dictionary<string,double> table)
       {
           double sum = 0;

           foreach (KeyValuePair<string, double> kv in table)
           {
               sum += kv.Value;
           }


           Dictionary<string, double> tfTable = new Dictionary<string,
double>();
           foreach (KeyValuePair<string, double> kv in table)
           {
               tfTable.Add(kv.Key, kv.Value / sum);
           }

           return tfTable;

       }

        private static Dictionary<string, double>
IDFDocumentTable(Dictionary<string, double>[] PreparedTables)
        {
            if (PreparedTables == null)
                throw new InvalidOperationException("Prepared Tables
can not be null..");
```

```
            Dictionary<string, double> tfidfTable = new
Dictionary<string, double>();

                        int di = 0;

                        int nDoc = PreparedTables.Length;


            foreach (KeyValuePair<string, double> kv in
PreparedTables[0])
            {
                foreach (Dictionary<string, double> table in
PreparedTables)
                {
                    if (table.ContainsKey(kv.Key))
                        di++;
                }
                tfidfTable.Add(kv.Key, (Math.Log(nDoc / di)) + 1);
                di = 0;
            }

            return tfidfTable;

        }

        private static Dictionary<string, double>[]
GetPreparedTFIDFTables(Dictionary<string, double> tfidfTable, params
Dictionary<string, double>[] Tables)
        {
            if (Tables == null)
                throw new InvalidOperationException("Tables can not be
null..");

            ArrayList tableList = new ArrayList();

            foreach (Dictionary<string, double> table in Tables)
            {
                Dictionary<string, double> newTable = new
Dictionary<string, double>();
                foreach (KeyValuePair<string, double> kv in table)
                {
                    newTable.Add(kv.Key, tfidfTable[kv.Key] *
kv.Value);
                }
                tableList.Add(newTable);
            }

            return (Dictionary<string,
double>[])tableList.ToArray(typeof(Dictionary<string, double>));

        }

        private static void PrepareAllHashTables(Dictionary<string,
double>[] tables)
        {
            for (int i = 0; i < tables.Length; i++)
            {
```

```
                for (int j = 1; j < tables.Length; j++)
                    PrepareTwoHashTable(tables[i], tables[j]);
            }
        }

        private static void PrepareTwoHashTable(Dictionary<string,
double> table1, Dictionary<string, double> table2)
        {
            //for table1
            foreach (KeyValuePair<string,double> kv in table1)
            {
                if (!table2.ContainsKey(kv.Key))
                    table2.Add(kv.Key, 0);
            }

            //for table2
            foreach (KeyValuePair<string, double> kv in table2)
            {
                if (!table1.ContainsKey(kv.Key))
                    table1.Add(kv.Key, 0);
            }

        }

        private static double CosineSimilarity(Dictionary<string,
double> table1, Dictionary<string, double> table2)
        {
            if (table1.Count != table2.Count)
                throw new InvalidOperationException("Table sizes must
be equal");

            //length of table 1
            double length1 = 0;
            double length2 = 0;

            //double firstValue;
            double secValue;

            //sum of vector multiplication
            double svMul = 0;

            foreach (KeyValuePair<string,double> kv in table1)
            {

                length1 += Math.Pow(kv.Value, 2);

                secValue = table2[kv.Key];
                length2 += Math.Pow(secValue,2);

                svMul += secValue * kv.Value;




            }

            length1 = Math.Sqrt(length1);
```

```
            length2 = Math.Sqrt(length2);

            return svMul / (length1 * length2);
        }
    }
    }
```

Appendix B

Xml_ReaperApp

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Xml;
using System.IO;
using System.Linq;
using System.Collections;
using System.Text.RegularExpressions;
using System.Data;
using System.Data.SqlClient;

namespace CompareXML
{

    class Program
    {

        static void Main(string[] args)
        {


            string path3 = @"C:\result\Myresult1.txt";
            string path4 = @"c:\MyTest4.txt";
            ArrayList same_item=new ArrayList();
            string result;
            path4 = Console.ReadLine();
            result = Path.GetFileNameWithoutExtension(path4 );
            Console.WriteLine( result);
            ArrayList arrText = new ArrayList();
            ArrayList list = new ArrayList();
            ArrayList list2 = new ArrayList();
            ArrayList list3 = new ArrayList();
            ArrayList list4 = new ArrayList();
            ArrayList name = new ArrayList();
            List<string> distinct_from_base = new List<string>();
            List<string> B = new List<string>();
            List<string> distinct_from_extern = new List<string>();
            List<string> C = new List<string>();
            List<string> val_of_union_b_and_c = new List<string>();
            List<double> S_value = new List<double>();
            List<double> base_of_sim1 = new List<double>();

            string[] name1 = new string[20];
            XmlDataDocument doc = new XmlDataDocument();
            //load the base source
            doc.Load("C:\\s5a.xml");
            XmlElement root = doc.DocumentElement;
            XmlNodeList elemList = root.GetElementsByTagName("RECORD");
```

```
            XmlTextReader reader2 = new XmlTextReader(path4);

            var nodes = doc.SelectNodes("S1/RECORD");


                // Create a file to write to.
            string file1 = path4;
                string file2 = @"C:\s5a.xml";

                if (File.Exists(file1) && File.Exists(file2))
                {
                    XmlTextReader rdrXml2 = new XmlTextReader(file2);
                    XmlTextReader rdrXml1 = new XmlTextReader(file1);

                    while (rdrXml2.Read())
                    {
                        switch (rdrXml2.NodeType)
                        {
                            case XmlNodeType.Element:
                                list2.Add(rdrXml2.Name);
                                list2.Remove("d5");
                                list2.Remove("RECORD");
                                list2.Remove("D5");

                                break;

                        }
                    }
                    while (rdrXml1.Read())
                    {
                        switch (rdrXml1.NodeType)
                        {
                            case XmlNodeType.Element:
                                list.Add(rdrXml1.Name);
                                list.Remove(result);
                                list.Remove("RECORD");

                                list.Remove( result.ToUpper());

                                break;

                        }

                    }


                    Hashtable ht = new Hashtable();

                    foreach (string item2 in list2)
                    {
                        ht[item2] = null;
                    }
                    ArrayList distinclist2 = new ArrayList(ht.Keys);
                    list2.Clear();
                    string[] name_from_base =
distinclist2.ToArray(typeof(string)) as string[];
                    Hashtable ht1 = new Hashtable();
```

```
                    foreach (string item in list)
                    {
                        ht1[item] = null;
                    }
                    ArrayList distincArray = new ArrayList(ht1.Keys);

                    list.Clear();
                    string[] name_from_external =
distincArray.ToArray(typeof(string)) as string[];



                    foreach (string ele in name_from_base)
                    {


                        if (name_from_external.Contains(ele))
                        {

                            name.Add(ele);

                        }

                    }


                string [] same_item1 = name.ToArray(typeof(string)) as
string[];




                    Console.WriteLine("base " +
name_from_base.Count());
                    Console.WriteLine("external " +
name_from_external.Count());
                    Console.WriteLine("inter {0} ",
same_item1.Count());

                    DataSet dataSet2 = new DataSet();
                    DataSet dataSet1 = new DataSet();
                    dataSet2.Clear();
                    dataSet1.Clear();
                    XmlReader st2 = new XmlTextReader(path4);
                    XmlReader st1 = new XmlTextReader("C:\\s5a.xml");

                    dataSet2.ReadXml(st2);
                    dataSet1.ReadXml(st1);

                    for (int i = 0, j = 0; j < same_item1.Length; i++,
j++)
                    {
                        if (same_item1[i] == same_item1[j])
                        {
                            foreach (DataRow a in
dataSet1.Tables[1].Rows)
                            {
```

```
                                list3.Add(a[same_item1[i]]);


                            }

                            foreach (string T in list3)
                            {
                                B.Add(T);
                            }

                            foreach (DataRow d in
dataSet2.Tables[1].Rows)
                            {

                                list4.Add(d[same_item1[j]]);
                            }

                        }

                        foreach (string S in list4)
                        {
                            C.Add(S);

                        }


                        list3.Clear();
                        list4.Clear();
                        distinct_from_base.AddRange(B.Distinct());
                        distinct_from_extern.AddRange(C.Distinct());
                        val_of_union_b_and_c =
distinct_from_base.Union(distinct_from_extern).ToList();
                        //ouroperation  S calulation
                        double RgivenA, P_A, P_R, P_RgivenA, S_val,
base_of_sim;
                        RgivenA = val_of_union_b_and_c.Count();
                        P_A = distinct_from_base.Count();
                        P_R = distinct_from_extern.Count();
                        P_RgivenA = (float)(1 / RgivenA) / (float)(1 /
P_A);
                        S_val = ((float)P_RgivenA) *
(Math.Log10((float)P_RgivenA / (1 / (float)P_R)));
                        S_value.Add((double )S_val);
                        base_of_sim = (Math.Log10((float)P_RgivenA / (1
/ (float)P_R)));
                        base_of_sim1.Add((double)base_of_sim);
                        same_item.Add(same_item1[i]);
                        B.Clear();
                        C.Clear();
                        double[] svalu = S_value.ToArray();

                        distinct_from_base.Clear();
                        distinct_from_extern.Clear();
                        val_of_union_b_and_c.Clear();

        // store S values to storage
```

```
        //clear()all lists used to  prepare it for next use


      //depending on S value(condition) :
   //extracting similar  content  from external xml datasource to final
distination
 string connectionString =

@"Server=DFRS8D4J\JAMAL1;Database=basesource;Trusted_Connection=true";
                 DataSet dat = new DataSet();
                 DataTable sourceData = new DataTable();
                sourceData = dataSet2.Tables[1];
                 // open the destination data
                 using (SqlConnection destinationConnection =
                       new SqlConnection(connectionString))
                 {
                // open the connection
  destinationConnection.Open();
   using (SqlBulkCopy bulkCopy =
                                  new
SqlBulkCopy(destinationConnection.ConnectionString))
     {
        for (int n = 0, m = 0; m < same_item1.Length; n++, m++)
         {

           bulkCopy.DestinationTableName = "base";


             // column mappings

bulkCopy.ColumnMappings.Add(same_item1[n], same_item1[m]);


      }

                         bulkCopy.WriteToServer(sourceData);
                    }
                 }
        }
    }


                //data extraction end


            using (StreamWriter rw = File.AppendText(path3))
           {
                rw.WriteLine("Feature Name                    S Value
Base of Similarity  ", true);
rw.WriteLine("-----------------------------------------------", true);
                for (int n = 0; n < S_value.Count(); n++)
                {
                    rw.WriteLine("  {0}                          {1}
{2} ", same_item[n], (float)S_value[n], base_of_sim1[n], true);
rw.WriteLine("-----------------------------------------------", true);
```

```
                }
                rw.Close();
            }
        }
        public static string CreateTABLE(string connectionString,
string tableName, DataTable table)
    {
        string sqlsc;
        using (SqlConnection connection = new
SqlConnection(connectionString))
        {
            connection.Open();
            sqlsc = "CREATE TABLE "+tableName+"(";
            for (int i = 0; i < table.Columns.Count; i++)
            {
                sqlsc += "\n [" + table.Columns[i].ColumnName + "] ";
                if
(table.Columns[i].DataType.ToString().Contains("System.Int32"))
                        sqlsc += " int ";
                else if
(table.Columns[i].DataType.ToString().Contains("System.DateTime"))
                        sqlsc += " datetime ";
                else if
(table.Columns[i].DataType.ToString().Contains("System.String"))
                    sqlsc += " nvarchar(" +
table.Columns[i].MaxLength.ToString() + ") ";
                else
                    sqlsc += " nvarchar(" +
table.Columns[i].MaxLength.ToString() + ") ";

                if (table.Columns[i].AutoIncrement)
                    sqlsc += " IDENTITY(" +
table.Columns[i].AutoIncrementSeed.ToString() + "," +
table.Columns[i].AutoIncrementStep.ToString() + ") ";
                if (!table.Columns[i].AllowDBNull)
                    sqlsc += " NOT NULL ";
                sqlsc += ",";
            }
            connection.Close();
        }
        return sqlsc.Substring(0,sqlsc.Length-1) + ")";
    }

    }
}
```