**The instructions on how to implement the DLL:**

Download the SACNASP\_DATA.dll.

Create the project on the C# .net Framework.

Once that has been created there is going to be a tab for references then right click to find the add references tab.

Click on add references then click on the Browse button at the bottom.

Find where you have stored the dll file, add it then press Ok.

Adding files:

* Add the ‘listOfConjuctions.txt’ and ‘listOfScieneTerms.txt’ to the “bin/debug” path
* For the university files I recommend you create a folder in the “bin/debug” path and put all the csv files inside it
* If you add the files incorrectly then the dll won’t work

At the Top of your project import the namespace by stating “using SACNASP\_DATA;”

Then create an instance of the “PercentageComputation” class as an attribute.

Now you are free to implement all the functions.

**Example Implementation:**



**Functions available to test( for Ozi):**

-static int ComputeLevenshteinDistance(string s, string t)

-public void setUniversuty(List<string> universityFiles)

-public void SortAndRemoveDuplicatesFromFile(string filepath)

- public string CurateText(string text, string filepath)

//Use "listOfScieneTerms.txt" provided in the 'Necessary Files' folder for filepath

**Code behind Implementation:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using SACNASP\_DATA;

namespace TestRegisterable

{

public partial class Form1 : Form

{

public string[] UniverityFiles = { "UniversityFiles\\Cape Peninsula University of Technology.csv", "UniversityFiles\\Central\_University\_of\_Technology.csv", "UniversityFiles\\FortHare.csv", "UniversityFiles\\Nelson\_Mandela\_University.csv",

"UniversityFiles\\Rhodes University.csv", "UniversityFiles\\Sefako Makgatho Health Sciences University.csv", "UniversityFiles\\Tshwane University of Technology.csv", "UniversityFiles\\UNISA.csv", "UniversityFiles\\University of Free State.csv","UniversityFiles\\University of Johannesburg.csv",

"UniversityFiles\\University of Mpumalanga.csv", "UniversityFiles\\University of Witwatersrand.csv", "UniversityFiles\\University\_of\_Cape\_Town.csv", "UniversityFiles\\University\_of\_limpopo.csv",

"UniversityFiles\\University\_of\_Pretoria.csv", "UniversityFiles\\University\_of\_Venda.csv", "UniversityFiles\\University\_of\_Western\_Cape.csv", "UniversityFiles\\Vaal University of Technology.csv", "UniversityFiles\\Walter Sizulu.csv"};

PercentageComputation percentageComputation = new PercentageComputation();

public Form1()

{

InitializeComponent();

percentageComputation.setUniversuty(UniverityFiles.ToList<string>());

string[] Universities = percentageComputation.getUniversities();

foreach (string University in Universities)

{

cb\_Universities.Items.Add(University);

}

}

private void btn\_Calculate\_Click(object sender, EventArgs e)

{

double score = 0;

if (tb\_OtherQualification.Text != "")

{

score = percentageComputation.OtherSelected(tb\_OtherQualification.Text);

}

else

{

score = percentageComputation.getQualificationPercentage(cb\_Qualifications.SelectedText);

}

score += percentageComputation.CalculationAdditions(Convert.ToInt32(tb\_NQFLevel.Text), Convert.ToInt32(tb\_YesrsOfExp.Text));

score = score / 3;

lbl\_Score.Text = score.ToString();

}

private void cb\_Universities\_SelectedIndexChanged(object sender, EventArgs e)

{

cb\_Qualifications.Items.Clear();

List<string> Qualifications = percentageComputation.getQualifications(cb\_Universities.SelectedIndex);

foreach(string qualification in Qualifications)

{

cb\_Qualifications.Items.Add(qualification);

}

}

private void cb\_Qualifications\_SelectedIndexChanged(object sender, EventArgs e)

{

}

}

}

**All the code inside the Dll in case you need it:**

using System;

using System.Collections.Generic;

using System.Data.Common;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Xml.Schema;

using Microsoft.VisualBasic.FileIO;

namespace SACNASP\_DATA

{

public class PercentageComputation

{

public static string inputText = "";

public static string outputText = "";

public static int universityIndex = 0;

public static List<string> QualificationNames = new List<string>();

public static List<int> QualificationIndexes = new List<int>();

public static List<string[]> list\_of\_fields = new List<string[]> { };

public static List<string> UniversityFiles = new List<string>();

public int checkForKeywords(string searchText, string filePath)

{

// Read the keywords from the file

string[] keywords = File.ReadAllLines(filePath);

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

// Split the search text into individual words

string[] searchWords = searchText.Split(' ');

int[,] distances = new int[keywords.Length, searchWords.Length];

// Count the number of matches between the keywords and the search text

int score = 0;

for (int i = 0; i < keywords.Length; i++)

{

for (int j = 0; j < searchWords.Length; j++)

{

distances[i, j] = ComputeLevenshteinDistance(keywords[i], searchWords[j]);

if (distances[i, j] <= 2)

{

score++;

}

//Console.WriteLine($"Distance between {scienceWords[i]} and {otherWords[j]} is {distances[i, j]}");

}

}

return score;

}

public void setInputText(string Text)

{

inputText = Text;

}

public string getOutputText()

{

return outputText;

}

private void setOutputText(string text)

{

outputText = text;

}

public void setUniversuty(List<string> universityFiles)

{

foreach (string Filename in universityFiles)

{

UniversityFiles.Add(Filename);

}

}

public double CalculatePercentaged(string inputText)

{

string ScoreLabel = "";

SortAndRemoveDuplicatesFromFile("listOfConjuctions.txt");

string filePath = "listOfScieneTerms.txt";

string curatedTxt = CurateText(inputText, "listOfConjuctions.txt");

setOutputText(curatedTxt);

int x = checkForKeywords(curatedTxt, filePath);

int y = curatedTxt.Split(' ').Length;

double score = Convert.ToDouble(x) / Convert.ToDouble(y) \* 100;

ScoreLabel = "Score: " + score.ToString("F2") + "%";

return score;

}

public void SortAndRemoveDuplicatesFromFile(string filepath)

{

// Read all lines from the file into an array

string[] lines = File.ReadAllLines(filepath);

// Sort the lines in alphabetical order

Array.Sort(lines);

// Create a HashSet to store the unique terms

HashSet<string> uniqueTerms = new HashSet<string>();

// Iterate through the sorted lines and add each unique term to the HashSet

foreach (string line in lines)

{

if (!uniqueTerms.Contains(line))

{

uniqueTerms.Add(line);

}

}

// Overwrite the file with the sorted, unique terms

using (StreamWriter writer = new StreamWriter(filepath))

{

foreach (string term in uniqueTerms)

{

writer.WriteLine(term);

}

}

}

public string CurateText(string text, string filepath)

{

string[] termsToRemove = File.ReadAllLines(filepath);

// Split the input text into words

string[] words = text.Split(' ');

// Remove the terms from the words

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

foreach (string word in words)

{

bool shouldRemove = false;

foreach (string termToRemove in termsToRemove)

{

if (word.Equals(termToRemove, StringComparison.OrdinalIgnoreCase))

{

shouldRemove = true;

break;

}

}

if (!shouldRemove)

{

curatedWords.Add(word);

}

}

// Join the curated words into a single string

string curatedText = string.Join(" ", curatedWords.ToArray());

return curatedText;

}

static int ComputeLevenshteinDistance(string s, string t)

{

if (string.IsNullOrEmpty(s))

{

if (string.IsNullOrEmpty(t))

return 0;

return t.Length;

}

if (string.IsNullOrEmpty(t))

{

return s.Length;

}

int n = s.Length;

int m = t.Length;

int[,] d = new int[n + 1, m + 1];

for (int i = 0; i <= n; d[i, 0] = i++) ;

for (int j = 0; j <= m; d[0, j] = j++) ;

for (int i = 1; i <= n; i++)

{

for (int j = 1; j <= m; j++)

{

int cost = (t[j - 1] == s[i - 1]) ? 0 : 1;

d[i, j] = Math.Min(

Math.Min(d[i - 1, j] + 1, d[i, j - 1] + 1),

d[i - 1, j - 1] + cost);

}

}

return d[n, m];

}

public static int SearchForKeywords(string text, string filePath)

{

int score = 0;

string[] termsToScore = File.ReadAllLines(filePath);

// Split the input text into words

string[] words = text.Split(' ');

// Remove the terms from the words

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

foreach (string word in words)

{

bool shouldScore = false;

foreach (string termToScore in termsToScore)

{

if (word.Equals(termToScore, StringComparison.OrdinalIgnoreCase))

{

shouldScore = true;

break;

}

}

if (shouldScore)

{

score++;

}

}

// Join the curated words into a single string

string curatedText = string.Join(" ", curatedWords.ToArray());

return score;

}

public string[] getUniversities()

{

list\_of\_fields.Clear();

string[] universityNames = new string[UniversityFiles.Count];

foreach (string university in UniversityFiles)

{

string[] split;

string[] nextSplit;

split = UniversityFiles[universityIndex].Split('\\');

nextSplit = split[split.Length-1].Split('.');

universityNames[universityIndex] = nextSplit[0].Replace('\_', ' ');

universityIndex++;

}

return universityNames;

}

public List<string> getFaculties()

{

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

if (universityIndex != -1)

{

foreach (string[] rows in list\_of\_fields)

{

if (rows[1] != "null")

{

if (!faculties.Contains(rows[1]))

{

faculties.Add(rows[1]);

}

}

}

}

return faculties;

}

public List<string> getQualifications(int UniversityIndex)

{

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

qualifications.Clear();

if (UniversityIndex != -1)

{

string filePath = UniversityFiles[UniversityIndex];

using (TextFieldParser parser = new TextFieldParser(filePath))

{

parser.TextFieldType = FieldType.Delimited;

parser.SetDelimiters(";");

while (!parser.EndOfData)

{

string[] fields = parser.ReadFields();

list\_of\_fields.Add(fields);

}

}

}

int rowCount = 0;

foreach (string[] rows in list\_of\_fields)

{

if (rows[2] != "null" && rows[2] != "none" && rows[2] != "N/A" && rows[2] != "n/a")

{

if (!qualifications.Contains(rows[2]))

{

QualificationIndexes.Add(rowCount);

QualificationNames.Add(rows[2]);

qualifications.Add(rows[2]);

}

}

rowCount++;

}

return qualifications;

}

public double getQualificationPercentage (string QualificationSelected)

{

string descriptions = "";

bool stopReading = false;

string QualificationName = "";

int startQualificationRecord = 0;

int endQualificationRecord = 0;

int recordCount = 1;

int qualificationIndex = 0;

foreach (string qualification in QualificationNames)

{

if (qualification == QualificationSelected)

{

startQualificationRecord = qualificationIndex;

try

{

endQualificationRecord = QualificationIndexes.IndexOf(qualificationIndex + 1);

}

catch(IndexOutOfRangeException ex)

{

endQualificationRecord = list\_of\_fields.Count;

}

}

qualificationIndex++;

}

foreach (string[] record in list\_of\_fields)

{

if (recordCount == startQualificationRecord)

{

QualificationName = record[2];

if (record[7] == "null" && record[7] == "none" && record[7] == "N/A" && record[7] == "n/a")

{

descriptions += " " + QualificationName;

}

}

else if (recordCount == endQualificationRecord)

{

stopReading = true;

}

else if (!stopReading)

{

descriptions += " " + record[7];

}

}

return CalculatePercentaged(descriptions);

}

public double OtherSelected(string OtherQualificationName)

{

return CalculatePercentaged(OtherQualificationName);

}

public double CalculationAdditions(int NQFLevel, int YearsOfExperience)

{

double scoreAddition = 0;

int[] validNQFLevels = { 8, 9, 10 };

if (validNQFLevels.Contains(NQFLevel))

{

if (NQFLevel == 8)

{

scoreAddition += 80;

}

else if (NQFLevel == 9)

{

scoreAddition += 90;

}

else if (NQFLevel == 10)

{

scoreAddition += 100;

}

}

if (YearsOfExperience == 1 || YearsOfExperience == 2)

{

scoreAddition += 20;

}

else if (YearsOfExperience == 3 || YearsOfExperience == 4)

{

scoreAddition += 40;

}

else if (YearsOfExperience == 5 || YearsOfExperience == 6)

{

scoreAddition += 60;

}

else if (YearsOfExperience == 7 || YearsOfExperience == 8)

{

scoreAddition += 80;

}

else if (YearsOfExperience == 3 || YearsOfExperience == 4)

{

scoreAddition += 100;

}

else if (YearsOfExperience > 10)

{

scoreAddition += 100;

}

return scoreAddition;

}

}

}