This is the part where the system label each word in the database

ServiceBehavior(AddressFilterMode = AddressFilterMode.Any)]

public class Service1 : IService1

private Dictionary<string, double> GetDictionary()

{

Dictionary<string, double> myDictionary = new Dictionary<string, double>();

try

{

SqlConnection con = new SqlConnection(@"Data Source=(localdb)\KRQPInstance;Initial Catalog=KRQP\_DB;Integrated Security=true;");

con.Open();

SqlCommand cmdH = new SqlCommand("USE [KRQP\_DB] select text,score from words", con);

//SqlDataReader drWords = cmdH.ExecuteReader();

string text = "";

double score = 0;

using (SqlDataReader reader = cmdH.ExecuteReader())

{

while (reader.Read())

{

text = reader["text"].ToString();

score = double.Parse(reader["score"].ToString());

myDictionary.Add(text, score);

}

reader.Close();

}

con.Close();

}

catch (Exception ex)

{

}

return myDictionary;

Where the system borrow and request the data from the twitter

public string Analyze(Request pa)

{

Response response = new Response();

try

{

response.DepressedWords = new List<string>();

response.PostiveWords = new List<string>();

var auth = new SingleUserAuthorizer

{

CredentialStore = new SingleUserInMemoryCredentialStore

{

ConsumerKey = "euyu2tF3avfOAQfGVZ08Vq2J01",

ConsumerSecret = "br7xPVRxQ1JBGSrusEZhwmYf3vGRslwSTJshGGFbmALhY0XDDu2",

AccessToken = "92449081-ut3HwkOlZjMw9wgTtkwikrEpuhLgtlwH4XdR8hcHP3",

AccessTokenSecret = "xqAkzeCpExLZBSEP9xslLy9thMCzAooiAb6nHEWxxreaQ4"

}

};

This is where the segregation happen by labeling the word with polarity score to positive and negative and classified if it is true or false. And sorting to the history graph data of the user tweeter account.

Dictionary<string, double> AllWords = GetDictionary();

//SEGREGATION NG MGA WORDS NG TWEET SA WORD SA DB

var problemBuilder = new TextClassificationProblemBuilder();

//PROBLEM IS THE SEGREGATED LIST

var problem = problemBuilder.CreateProblem(AllWords.Keys, AllWords.Values.ToArray(), AllWords.Keys.ToArray());

const int c = 1;

C\_SVC model = new C\_SVC(problem, KernelHelper.LinearKernel(), c);

var twitterCtx = new TwitterContext(auth);

var tweets =

(from tweet in twitterCtx.Status

where tweet.Type == StatusType.User

&& tweet.ScreenName == pa.userName

select tweet).ToList().Take(pa.count).ToList();

List<Tweet> tweetList = new List<Tweet>();

foreach (var tweet in tweets)

{

Tweet tempTweet = new Tweet();

tempTweet.Text = tweet.Text;

tempTweet.Date = tweet.CreatedAt.Date.ToShortDateString();

tweetList.Add(tempTweet);

}

foreach (Tweet tweet in tweetList)

{

double collectiveScore = 0;

double positiveScore = 0;

double negativeScore = 0;

int sentimentCount = 0;

List<string> splitTweet = new List<string>(tweet.Text.Split(' '));

if (splitTweet.Contains("not"))

{

//positiveScore = -2;

negativeScore += 1;

sentimentCount++;

}

bool isLastWordNot = false;

foreach (string word in splitTweet)

{

if (word == "not")

{

isLastWordNot = true;

}

else

{

//

var newX = TextClassificationProblemBuilder.CreateNode(word, AllWords.Keys.ToArray());

double val = model.Predict(newX);

if (val < 0 && newX.Count() > 0)

{

response.DepressedWords.Add(word);

if (isLastWordNot)

{

negativeScore = negativeScore - 2;

isLastWordNot = false;

}

negativeScore = negativeScore + 1;

sentimentCount++;

//collectiveScore = collectiveScore - 1;

//collectiveScore = collectiveScore - val;

}

else if (val > 0)

{

response.PostiveWords.Add(word);

if (isLastWordNot)

{

positiveScore = positiveScore - 2;

isLastWordNot = false;

}

//collectiveScore = collectiveScore + 1;

sentimentCount++;

positiveScore = positiveScore + 1;

//collectiveScore = collectiveScore + val;

}

}

}

collectiveScore = positiveScore - negativeScore - (0.5 \* sentimentCount);

if (collectiveScore < 0)

{

tweet.isDepressed = true;

}

else

{

tweet.isDepressed = false;

}

}

This part is the computation is happening and decide if it positive, neutral and stressed.

response.Tweets = tweetList;

int score = (int)Math.Ceiling((decimal)((decimal)tweetList.Where(x => x.isDepressed == true).Count() / (decimal)tweetList.Count()) \* 100);

if (score <= 30)

{

response.Emotion = "Postive";

}

else if (score > 30 && score <= 60)

{

response.Emotion = "Neutral";

}

else if (score > 60)

{

response.Emotion = "Stressed";

}

response.Percentage = score.ToString();

SqlConnection con = new SqlConnection(@"Data Source=(localdb)\KRQPInstance;Initial Catalog=KRQP\_DB;Integrated Security=true;");

con.Open();

string query = "USE [KRQP\_DB] insert into history (username,percentage) values ('" + pa.userName + "','" + score.ToString() + "')";

SqlCommand cmd = new SqlCommand(query, con);

int res = cmd.ExecuteNonQuery();

con.Close();

}

catch (Exception ex)

{

response.Error = ex.Message;

}

return JsonConvert.SerializeObject(response);

}

public string Analyze1(Request pa)

{

return "";

}

Where the graph manage of its username and their percentage in a graph

public string History(Request pa)

{

HistoryReponse historyRepsonse = new HistoryReponse();

historyRepsonse.TopRows = new List<Response>();

try

{

SqlConnection con = new SqlConnection(@"Data Source=(localdb)\KRQPInstance;Initial Catalog=KRQP\_DB;Integrated Security=true;");

con.Open();

SqlCommand cmdH = new SqlCommand("USE[KRQP\_DB] select top 20 username,percentage from history order by id desc", con);

using (SqlDataReader reader = cmdH.ExecuteReader())

{

while (reader.Read())

{

Response response = new Response();

response.Username = reader["username"].ToString();

response.Percentage = reader["percentage"].ToString();

historyRepsonse.TopRows.Add(response);

}

reader.Close();

}

con.Close();

}

catch (Exception ex)

{

}

return JsonConvert.SerializeObject(historyRepsonse);

}

public string History1(Request pa)

{

return "";

}

This process where you can add words to the database

public string Words(Request pa)

{

Response response = new Response();

response.AllWords = new List<Word>();

try

{

SqlConnection con = new SqlConnection(@"Data Source=(localdb)\KRQPInstance;Initial Catalog=KRQP\_DB;Integrated Security=true;");

con.Open();

SqlCommand cmdH = new SqlCommand("USE[KRQP\_DB] select text,score from words order by text", con);

using (SqlDataReader reader = cmdH.ExecuteReader())

{

while (reader.Read())

{

Word word = new Word();

word.Text = reader["text"].ToString();

word.Score = int.Parse(reader["score"].ToString());

response.AllWords.Add(word);

}

reader.Close();

}

con.Close();

}

catch (Exception ex)

{

}

return JsonConvert.SerializeObject(response);

}

public string Words1(Request pa)

{

return "";

}

This process is the log in page where you need to log in before you can edit the words in the database

public string Login(Request pa)

{

SqlConnection con = new SqlConnection(@"Data Source=(localdb)\KRQPInstance;Initial Catalog=KRQP\_DB;Integrated Security=true;");

con.Open();

Response response = new Response();

try

{

SqlCommand cmdH = new SqlCommand("USE [KRQP\_DB] select username from admins where username = '" + pa.userName + "' and password = '" + pa.password + "';", con);

SqlDataReader drLogin = cmdH.ExecuteReader();

string username = "";

while (drLogin.Read())

{

username = drLogin["username"].ToString();

}

drLogin.Close();

if (username != string.Empty)

{

response.LoginSuccessful = true;

}

else

{

response.LoginSuccessful = false;

}

con.Close();

}

catch (Exception ex)

{

response.Error = ex.Message;

}

return JsonConvert.SerializeObject(response); ;

}

public string Login1(Request pa)

{

return "";

}

A sample code which from hard coded words you can import it to the database

public string Import(Request pa)

{

SqlConnection con = new SqlConnection(@"Data Source=(localdb)\KRQPInstance;Initial Catalog=KRQP\_DB;Integrated Security=true;");

con.Open();

Response response = new Response();

Try

tring query = "use [KRQP\_DB] insert into words (text,score) values(');<', -2.6);use [KRQP\_DB] insert into words (text,score)

SqlCommand cmd = new SqlCommand(query, con);

int res = cmd.ExecuteNonQuery();

response.Error = "Words Imported Successfully";

con.Close();

}

catch (Exception ex)

{

response.Error = ex.Message;

}

return JsonConvert.SerializeObject(response); ;

}

public string Import1(Request pa)

{

return "";

}

This is how the system add words to the database

public string AddWord(WordRequest pa)

{

SqlConnection con = new SqlConnection(@"Data Source=(localdb)\KRQPInstance;Initial Catalog=KRQP\_DB;Integrated Security=true;");

con.Open();

Response response = new Response();

try

{

SqlCommand cmdH = new SqlCommand("USE [KRQP\_DB] select text from words where text = '" + pa.text + "' and score = " + int.Parse(pa.score) + ";", con);

SqlDataReader check = cmdH.ExecuteReader();

string username = "";

while (check.Read())

{

username = check["text"].ToString();

}

check.Close();

if (username == string.Empty)

{

string query = "USE [KRQP\_DB] insert into words (text,score) values ('" + pa.text + "'," + pa.score + ")";

SqlCommand cmd = new SqlCommand(query, con);

int res = cmd.ExecuteNonQuery();

response.Error = pa.text + " Added Successfully with polarity score " + pa.score;

}

else

{

response.Error = pa.text + " already exists with polarity score " + pa.score;

}

con.Close();

}

catch (Exception ex)

{

response.Error = ex.Message;

}

return JsonConvert.SerializeObject(response); ;

}

public string AddWord1(WordRequest pa)

{

return "";

}

delete words from the database

public string DeleteWord(WordRequest pa)

{

SqlConnection con = new SqlConnection(@"Data Source=(localdb)\KRQPInstance;Initial Catalog=KRQP\_DB;Integrated Security=true;");

con.Open();

Response response = new Response();

try

{

SqlCommand cmdH = new SqlCommand("USE [KRQP\_DB] select text from words where text = '" + pa.text + "' and score = " + int.Parse(pa.score) + ";", con);

SqlDataReader check = cmdH.ExecuteReader();

string username = "";

while (check.Read())

{

username = check["text"].ToString();

}

check.Close();

if (username != string.Empty)

{

string query = "USE [KRQP\_DB] delete from words where text = '" + pa.text + "' and score = " + int.Parse(pa.score) + ";";

SqlCommand cmd = new SqlCommand(query, con);

int res = cmd.ExecuteNonQuery();

response.Error = pa.text + " Deleted Successfully with polarity score " + pa.score;

}

else

{

response.Error = pa.text + " does not exists with polarity score " + pa.score;

}

con.Close();

}

catch (Exception ex)

{

response.Error = ex.Message;

}

return JsonConvert.SerializeObject(response); ;

}

public string DeleteWord1(WordRequest pa)

{

return "";

}

Modify words from the database

public string ModifyWord(WordRequest pa)

{

SqlConnection con = new SqlConnection(@"Data Source=(localdb)\KRQPInstance;Initial Catalog=KRQP\_DB;Integrated Security=true;");

con.Open();

Response response = new Response();

try

{

SqlCommand cmdH = new SqlCommand("USE [KRQP\_DB] select text from words where text = '" + pa.text + "' and score = " + int.Parse(pa.score) + ";", con);

SqlDataReader check = cmdH.ExecuteReader();

string username = "";

while (check.Read())

{

username = check["text"].ToString();

}

check.Close();

if (username != string.Empty)

{

string query = "USE [KRQP\_DB] update words set text = '" + pa.newText + "', score = " + int.Parse(pa.newScore) + " where text = '" + pa.text + "' and score = " + int.Parse(pa.score) + ";";

SqlCommand cmd = new SqlCommand(query, con);

int res = cmd.ExecuteNonQuery();

response.Error = pa.newText + " Modified Successfully with polarity score " + pa.newScore;

}

else

{

response.Error = pa.text + " does not exists with polarity score " + pa.score;

}

con.Close();

}

catch (Exception ex)

{

response.Error = ex.Message;

}

return JsonConvert.SerializeObject(response); ;

}

public string ModifyWord1(WordRequest pa)

{

return "";

}

Deleting all words in just 1 click in the database

public string Clear(Request pa)

{

SqlConnection con = new SqlConnection(@"Data Source=(localdb)\KRQPInstance;Initial Catalog=KRQP\_DB;Integrated Security=true;");

con.Open();

Response response = new Response();

try

{

string query = "USE [KRQP\_DB] delete from words";

SqlCommand cmd = new SqlCommand(query, con);

int res = cmd.ExecuteNonQuery();

response.Error = "All Words Deleted Successfully";

con.Close();

}

catch (Exception ex)

{

response.Error = ex.Message;

}

return JsonConvert.SerializeObject(response); ;

}

public string Clear1(Request pa)

{

return "";

}

}

}

The Text Classification builder where the SVM works

using libsvm;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Twitter

{

public class TextClassificationProblemBuilder

{

public svm\_problem CreateProblem(IEnumerable<string> x, double[] y, IReadOnlyList<string> vocabulary)

{

return new svm\_problem

{

y = y,

x = x.Select(xVector => CreateNode(xVector, vocabulary)).ToArray(),

l = y.Length

};

}

public static svm\_node[] CreateNode(string x, IReadOnlyList<string> vocabulary)

{

var node = new List<svm\_node>(vocabulary.Count);

string[] words = x.Split(new[] { ' ', '\t' }, StringSplitOptions.RemoveEmptyEntries);

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

{

int occurenceCount = words.Count(s => String.Equals(s, vocabulary[i], StringComparison.OrdinalIgnoreCase));

if (occurenceCount == 0)

continue;

node.Add(new svm\_node

{

index = i + 1,

value = occurenceCount

});

}

return node.ToArray();

}

}

}

“The history response”

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Twitter

{

public class HistoryReponse

{

public List<Response> TopRows { get; set; }

}

}

“The service 1”

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Serialization;

using System.ServiceModel;

using System.ServiceModel.Web;

using System.Text;

namespace Twitter

{

// NOTE: You can use the "Rename" command on the "Refactor" menu to change the interface name "IService1" in both code and config file together.

[ServiceContract]

public interface IService1

{

[OperationContract]

[WebInvoke(UriTemplate = "Analyze",

RequestFormat = WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "POST")]

string Analyze(Request dataJson);

[OperationContract]

[WebInvoke(UriTemplate = "Analyze",

RequestFormat = WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "OPTIONS")]

string Analyze1(Request dataJson);

[OperationContract]

[WebInvoke(UriTemplate = "Words",

RequestFormat = WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "POST")]

string Words(Request dataJson);

[OperationContract]

[WebInvoke(UriTemplate = "Words",

RequestFormat = WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "OPTIONS")]

string Words1(Request dataJson);

[OperationContract]

[WebInvoke(UriTemplate = "History",

RequestFormat = WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "POST")]

string History(Request dataJson);

[OperationContract]

[WebInvoke(UriTemplate = "History",

RequestFormat = WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "OPTIONS")]

string History1(Request dataJson);

[OperationContract]

[WebInvoke(UriTemplate = "Login",

RequestFormat = WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "POST")]

string Login(Request dataJson);

[OperationContract]

[WebInvoke(UriTemplate = "Login",

RequestFormat = WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "OPTIONS")]

string Login1(Request dataJson);

[OperationContract]

[WebInvoke(UriTemplate = "Import",

RequestFormat = WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "POST")]

string Import(Request dataJson);

[OperationContract]

[WebInvoke(UriTemplate = "Import",

RequestFormat = WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "OPTIONS")]

string Import1(Request dataJson);

[OperationContract]

[WebInvoke(UriTemplate = "Clear",

RequestFormat = WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "POST")]

string Clear(Request dataJson);

[OperationContract]

[WebInvoke(UriTemplate = "Clear",

RequestFormat = WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "OPTIONS")]

string Clear1(Request dataJson);

[OperationContract]

[WebInvoke(UriTemplate = "AddWord",

RequestFormat = WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "POST")]

string AddWord(WordRequest dataJson);

[OperationContract]

[WebInvoke(UriTemplate = "AddWord",

RequestFormat = WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "OPTIONS")]

string AddWord1(WordRequest dataJson);

[OperationContract]

[WebInvoke(UriTemplate = "DeleteWord",

RequestFormat = WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "POST")]

string DeleteWord(WordRequest dataJson);

[OperationContract]

[WebInvoke(UriTemplate = "DeleteWord",

RequestFormat = WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "OPTIONS")]

string DeleteWord1(WordRequest dataJson);

[OperationContract]

[WebInvoke(UriTemplate = "ModifyWord",

RequestFormat = WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "POST")]

string ModifyWord(WordRequest dataJson);

[OperationContract]

[WebInvoke(UriTemplate = "ModifyWord",

RequestFormat = WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "OPTIONS")]

string ModifyWord1(WordRequest dataJson);

}

}

“The Request”

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Serialization;

using System.Web;

namespace Twitter

{

[DataContract]

public class Request

{

[DataMember]

public string userName;

[DataMember]

public int count;

[DataMember]

public string password;

public Request()

{

userName = "";

count = 10;

password = "";

}

The response

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Twitter

{

public class Response

{

public List<Tweet> Tweets { get; set; }

public List<string> DepressedWords { get; set; }

public List<string> PostiveWords { get; set; }

public string Emotion { get; set; }

public string Error { get; set; }

public string Percentage { get; set; }

public bool LoginSuccessful { get; set; }

public string Username { get; set; }

public List<Word> AllWords {get; set;}

}

}

Set and get the tweet

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Twitter

{

public class Tweet

{

public string Date { get; set; }

public string Text { get; set; }

public bool isDepressed { get; set; }

}

}

Set and get the words

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Twitter

{

public class Word

{

public string Text { get; set; }

public int Score { get; set; }

}

}

Requesting words

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Serialization;

using System.Web;

namespace Twitter

{

[DataContract]

public class WordRequest

{

[DataMember]

public string text;

[DataMember]

public string score;

[DataMember]

public string newText;

[DataMember]

public string newScore;

public WordRequest()

{

text = "";

score = "";

newText = "";

newScore = "";

}

}

}