Reference

Becheikh, N., Ziam, S., Idrissi, O., Castonguay, Y., Landry, R., 2010. How to improve knowledge transfer strategies and practices in education? Answers from a systematic literature review. Res. High. Educ. J. 7, 1.

Dabrowski, M., 2013. Business Intelligence In Call Centers. Int. J. Comput. Inf. Technol. 2.

DobriSek, S., Gros, J., Mihelic, F., Pavesic, N., 1999. HOMER: a voice-driven text-to-speech system for the blind, in: Industrial Electronics, 1999. ISIE'99. Proceedings of the IEEE International Symposium on. IEEE, pp. 205–208.

Evgeniou, T., Cartwright, P., 2005. Barriers to Information Management. Eur. Manag. J. 23, 293–299. doi:10.1016/j.emj.2005.04.007

Fitzsimmons, J., 2005. Service Management: Operations, Strategy, Information Technology with Student CD, 5 edition. ed. McGraw-Hill/Irwin, Boston.

Gans, N., Koole, G., Mandelbaum, A., 2003. Telephone call centers: Tutorial, review, and research prospects. Manuf. Serv. Oper. Manag. 5, 79–141.

Garcia, D., Archer, T., Moradi, S., Ghiabi, B., 2012. Waiting in Vain: Managing Time and Customer Satisfaction at Call Centers. Psychology 3, 213–216. doi:10.4236/psych.2012.32030

Khan, R.A., Quadri, S.M.K., 2012. Business intelligence: an integrated approach. Bus. Intell. J. 5, 64–70.

Kotsovos, A., Kriemadis, A., n.d. CUSTOMER RELATIONSHIP MANAGEMENT (CRM) ON THE INTERNET-EVIDENCE FROM THE FOOTBALL SECTOR.

Kumar, A., Telang, R., 2012. Does the Web Reduce Customer Service Cost? Empirical Evidence from a Call Center. Inf. Syst. Res. 23, 721–737. doi:10.1287/isre.1110.0390

L. Michelle Bobbitt, Pratibha A. Dabholkar, 2001. Integrating attitudinal theories to understand and predict use of technology-based self-service: The Internet as an illustration. Int. J. Serv. Ind. Manag. 12, 423–450. doi:10.1108/EUM0000000006092

Larsson, A.O., Hrastinski, S., 2011. Blogs and blogging: Current trends and future directions. First Monday 16.

Lee, C.-S., Lee, C.C., Kwon, H.-B., 2007. A Framework of Outsourcing Decision-Making for Human Resource Information Systems. MIS Res. Relev. IT Pract. Korea Soc. Manag. Inf. Syst. Seoul Korea 1–6.

Lee, L.–., Oun-Young, M., 1988. Voice and text messaging-a concept to integrate the services of telephone and data networks, in: Communications, 1988. ICC'88. Digital Technology-Spanning the Universe. Conference Record., IEEE International Conference on. IEEE, pp. 408–412.

McGuire, K.A., Kimes, S.E., Lynn, M., Pullman, M.E., Lloyd, R.C., 2010. A framework for evaluating the customer wait experience. J. Serv. Manag. 21, 269–290.

Medhat, W., Hassan, A., Korashy, H., 2014. Sentiment analysis algorithms and applications: A survey. Ain Shams Eng. J. 5, 1093–1113. doi:10.1016/j.asej.2014.04.011

Meuter, M.L., Bitner, M.J., Ostrom, A.L., Brown, S.W., 2005. Choosing among alternative service delivery modes: An investigation of customer trial of self-service technologies. J. Mark. 69, 61–83.

Mozaheb, A., Alamolhodaei, S.M.A., Ardakani, M.F., others, 2015. Effect of customer relationship management (CRM) on performance of small-medium sized enterprises (SMEs) using structural equations model (SEM). Int. J. Acad. Res. Account. Finance Manag. Sci. 5, 42–52.

Oodith, D., Parumasur, S.B., 2014. Technology in a call center: an asset to managing customers and their needs? Probl. Perspect. Manag. 12.

Rasooli, P., Albadvi, A., 2007. Knowledge Management in Call Centres. Electron. J. Knowl. Manag. 5, 323–332.

Sabherwal, R., Becerra-Fernandez, I., 2011. Business intelligence : practices, technologies and management. Hoboken, NJ : Wiley.

Tsoukas, H., Vladimirou, E., 2001. What is Organizational Knowledge? J. Manag. Stud. 38, 973–993. doi:10.1111/1467-6486.00268

Tuerk, C., Monaco, P., Robinson, T., 1991. The development of a connectionist multiple-voice text-to-speech system, in: Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on. IEEE, pp. 749–752.

Appendix A

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Speech.Recognition;
using SpeechToText;
using System.IO;
using System.Diagnostics;
using System.Runtime.InteropServices;
using ADDIProtocol;
using NAudio.Wave;
using System.Threading;
using System.Collections;
using NetSpell.SpellChecker;
using System.Text.RegularExpressions;
using Markov;

namespace MSC
{
    public partial class Form1 : Form
    {

        #region locals

        /// <summary>
        /// data base util
        /// </summary>
        DB d = new DB();

        /// <summary>
        /// Markov Chain
        /// </summary>
        Chain<string> chain = null;

        /// <summary>
        /// Markov Chain initial words
        /// </summary>
        ArrayList chainstart = new ArrayList();

        /// <summary>
        /// prev recognized word from Markov Chain
        /// </summary>
        string prevrecognized = "";

        /// <summary>
        /// debug flag
        /// </summary>
        bool debug = true;

        /// <summary>
        /// Sites list for hot keys
        /// </summary>
        ArrayList sites = new ArrayList();

        /// <summary>
        /// SpellChecker
        /// </summary>
        Spelling spellChecker = new Spelling();

        /// <summary>
        /// Clustering
        /// </summary>
        BayesClassifier.Classifier m_Classifier = new BayesClassifier.Classifier();
        ArrayList vm_Classifier = new ArrayList();

        /// <summary>
        /// NAudio source
        /// </summary>
        private WaveIn waveSource = null;

        /// <summary>
        /// LowPass filter
        /// </summary>
        BiQuadFilter filter = null;

        /// <summary>
        /// WAV file writer
        /// </summary>
        public WaveFileWriter waveFile = null;

        /// <summary>
        /// the engine
        /// </summary>
        SpeechRecognitionEngine speechRecognitionEngine = null;

        /// <summary>
        /// CMDs
        /// </summary>
        Dictionary<string, string> dic = new Dictionary<string, string>();

        /// <summary>
        /// Key Registration
        /// </summary>
        private GlobalHotkey ghk1;
        private GlobalHotkey ghk2;
        private GlobalHotkey ghk3;
        private GlobalHotkey ghk4;
        private GlobalHotkey ghk5;
        private GlobalHotkey ghk6;

        #endregion

        #region invoke methods
        [DllImport("user32.dll")]
        static extern bool SetForegroundWindow(IntPtr hWnd);

        [DllImport("user32.dll")]
        private static extern int ShowWindow(IntPtr hwnd, int nCmdShow);

        [DllImport("user32.DLL", CharSet = CharSet.Unicode)]
        public static extern IntPtr FindWindow(String lpClassName, String lpWindowName);

        private const int SW_HIDE = 3;
        #endregion

        #region constructer
        /// <summary>
        /// Constructer
        /// </summary>
```

```
        public Form1()
        {

            InitializeComponent();
            Populatecluster();
            loadSites();
            LoadMarkov();

            try
            {
                // create the engine
                speechRecognitionEngine = createSpeechEngine("en-US");

                // hook to events
                speechRecognitionEngine.AudioLevelUpdated += new
EventHandler<AudioLevelUpdatedEventArgs>(engine_AudioLevelUpdated);
                speechRecognitionEngine.SpeechRecognized += new
EventHandler<SpeechRecognizedEventArgs>(engine_SpeechRecognized);
                speechRecognitionEngine.SpeechRecognitionRejected += new
EventHandler<SpeechRecognitionRejectedEventArgs>(recognizer_SpeechRecognitionRejected);

                // load dictionary
                loadGrammarAndCommands();
                loadCommands();

                if (debug)
                {
                    // use the system's default microphone
                    speechRecognitionEngine.SetInputToDefaultAudioDevice();
                    // start listening
                    speechRecognitionEngine.RecognizeAsync(RecognizeMode.Multiple);
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Voice recognition failed");
            }

            ghk1 = new GlobalHotkey(Constants.SHIFT, Keys.D1, this);
            ghk2 = new GlobalHotkey(Constants.SHIFT, Keys.D2, this);
            ghk3 = new GlobalHotkey(Constants.SHIFT, Keys.A, this);
            ghk4 = new GlobalHotkey(Constants.SHIFT, Keys.S, this);
            ghk5 = new GlobalHotkey(Constants.SHIFT, Keys.D, this);
            ghk6 = new GlobalHotkey(Constants.SHIFT, Keys.D3, this);

            try
            {
                // add event handlers
                spellChecker.MisspelledWord += new
NetSpell.SpellChecker.Spelling.MisspelledWordEventHandler(SpellChecker_MisspelledWord);
                spellChecker.EndOfText += new
NetSpell.SpellChecker.Spelling.EndOfTextEventHandler(SpellChecker_EndOfText);
                spellChecker.DoubledWord += new
NetSpell.SpellChecker.Spelling.DoubledWordEventHandler(SpellChecker_DoubledWord);
            }
            catch (Exception e)
            {

            }
        }

        #endregion
```

```
    #region internal functions and methods for speech recognition

    /// <summary>
    /// Creates the speech engine.
    /// </summary>
    /// <param name="preferredCulture">The preferred culture.</param>
    /// <returns></returns>
    private SpeechRecognitionEngine createSpeechEngine(string preferredCulture)
    {
      foreach (RecognizerInfo config in SpeechRecognitionEngine.InstalledRecognizers())
      {
        if (config.Culture.ToString() == preferredCulture)
        {
          speechRecognitionEngine = new SpeechRecognitionEngine(config);
          break;
        }
      }

      // if the desired culture is not found, then load default
      if (speechRecognitionEngine == null)
      {
        MessageBox.Show("The desired culture is not installed on this machine, the speech-engine will
continue using "
          + SpeechRecognitionEngine.InstalledRecognizers()[0].Culture.ToString() + " as the default
culture.",
          "Culture " + preferredCulture + " not found!");
        speechRecognitionEngine = new
SpeechRecognitionEngine(SpeechRecognitionEngine.InstalledRecognizers()[0]);
      }

      return speechRecognitionEngine;
    }


    /// <summary>
    /// Loads the commands.
    /// </summary>
    private void loadCommands()
    {
      try
      {
        string[] lines = File.ReadAllLines(Environment.CurrentDirectory + "\\CMD.txt");
        foreach (string line in lines)
        {
          // split the line
          var parts = line.Split(new char[] { '|' });
          dic[parts[0]] = parts[1];
        }
      }
      catch (Exception ex)
      {
        throw ex;
      }
    }

    /// <summary>
    /// Loads the grammar and commands.
    /// </summary>
    private void loadGrammarAndCommands()
    {
      try
      {
        Choices texts = new Choices();
```

```
                string[] lines = File.ReadAllLines(Environment.CurrentDirectory + "\\GRAM.txt");
                foreach (string line in lines)
                {
                    // skip commentblocks and empty lines..
                    if (line.StartsWith("--") || line == String.Empty) continue;

                    // split the line
                    var parts = line.Split(new char[] { '|' });

                    // add the text to the known choices of speechengine
                    texts.Add(parts[0]);
                }

                // Grammar
                Grammar wordsList = new Grammar(new GrammarBuilder(texts));
                speechRecognitionEngine.LoadGrammar(wordsList);

                // Create a default dictation grammar.
                DictationGrammar defaultDictationGrammar = new DictationGrammar();
                defaultDictationGrammar.Name = "default dictation";
                defaultDictationGrammar.Enabled = true;

                // Create the spelling dictation grammar.
                //DictationGrammar spellingDictationGrammar = new
DictationGrammar("grammar:dictation#spelling");
                //spellingDictationGrammar.Name = "spelling dictation";
                //spellingDictationGrammar.Enabled = true;

                //// Create the question dictation grammar.
                //DictationGrammar customDictationGrammar = new DictationGrammar("grammar:dictation");
                //customDictationGrammar.Name = "question dictation";
                //customDictationGrammar.Enabled = true;

                speechRecognitionEngine.LoadGrammar(defaultDictationGrammar);
                //speechRecognitionEngine.LoadGrammar(spellingDictationGrammar);
                //speechRecognitionEngine.LoadGrammar(customDictationGrammar);
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }

        #endregion

        #region speechEngine events

        /// <summary>
        /// Handles the SpeechRecognized event of the engine control.
        /// </summary>
        /// <param name="sender">The source of the event.</param>
        /// <param name="e">The <see cref="System.Speech.Recognition.SpeechRecognizedEventArgs"/>
instance containing the event data.</param>
        void engine_SpeechRecognized(object sender, SpeechRecognizedEventArgs e)
        {
            // This is for removing background noises
            if (e.Result.Confidence >= 0.2)
            {
                // recognized
                string recognized = e.Result.Text;
                string recognized1 = recognized;
                label3.Text = recognized;
                if (!textBox1.Text.Trim().Equals(""))
                {
                    // generate a new sequence using a starting word, and maximum return size
                    List<string> generated = new List<string>(chain.Generate(prevrecognized, 5));
                    IEnumerable<string> t = chain.Generate1(prevrecognized, 3);
                    int max = -1;

                    for (int i = 0; i < t.Count(); i++)
                    {
                        string ww = t.ElementAt(i);
                        double distance = CalculateSimilarity(ww, recognized);
                        int dis = (int)(distance * 100);
                        if (max < dis)
                        {
                            recognized1 = ww;
                            max = dis;
                        }
                    }
                }
                else
                {
                    // initial words
                    int max = -1;
                    for (int i = 0; i < chainstart.Count; i++)
                    {
                        string ww = (string)chainstart[i];
                        double distance = CalculateSimilarity(ww, recognized);
                        int dis = (int)(distance * 100);
                        if (max < dis)
                        {
                            recognized1 = ww;
                            max = dis;
                        }
                    }
                }

                prevrecognized = recognized1;
                textBox1.Text = (textBox1.Text.Trim() == "") ? recognized1 : textBox1.Text + "," + recognized1;
            }
        }

        /// <summary>
        /// Handles the AudioLevelUpdated event of the engine control.
        /// </summary>
        /// <param name="sender">The source of the event.</param>
        /// <param name="e">The <see cref="System.Speech.Recognition.AudioLevelUpdatedEventArgs"/>
instance containing the event data.</param>
        void engine_AudioLevelUpdated(object sender, AudioLevelUpdatedEventArgs e)
        {
            progressBar1.Value = e.AudioLevel;
        }

        /// <summary>
        /// Handle rejected words
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        void recognizer_SpeechRecognitionRejected(object sender, SpeechRecognitionRejectedEventArgs e)
        {
            string temp = "";
            foreach (RecognizedPhrase phrase in e.Result.Alternates)
            {
                temp = temp + "," + phrase.Text;
```

```
    }
    label3.Text = temp;
}



#endregion

#region HandleHotkey
/// <summary>
/// Execution of the CMD
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button1_Click(object sender, EventArgs e)
{

    // Classify data ------------------
    String tabName2 = textBox1.Text;
    String data = tabName2.Replace(",", " ");
    String temp = "";
    foreach (BayesClassifier.Classifier value in vm_Classifier)
    {
        temp = temp + "--->" + Classify(data, value);
    }
    label2.Text = temp;

    // Add to SQL DB for future reporting purposes
    d.Add(temp + "###" + DateTime.Now);
    // --------------------------------

    string cmd = comboBox1.Text;
    String tabName1 = textBox1.Text;
    String tabName = tabName1;
    if (dic.ContainsKey(tabName1))
    {
        tabName = dic[tabName1];
    }

    if (cmd.Equals("Execute"))
    {
        // Action
        IEAccessible ie = new IEAccessible();

        Process[] processes = Process.GetProcessesByName("iexplore");
        foreach (var item in processes)
        {
            ShowWindow(item.MainWindowHandle, SW_HIDE);
            SetForegroundWindow(item.MainWindowHandle);
        }

        // --
        SHDocVw.ShellWindows shellWindows = new SHDocVw.ShellWindows();
        foreach (SHDocVw.InternetExplorer window in shellWindows)
        {
            ie.Focus((IntPtr)window.HWND, tabName);
        }
        // --
    }
    if (cmd.Equals("Search"))
    {
        string prefix = "http://172.25.37.187/wordpress/?s=";
        string test = prefix + tabName.Replace(",", "+");
```

```
                string test1 = test + "+" + comboBox2.Text + "+" + comboBox3.Text + "+" + comboBox4.Text +
"+" + comboBox5.Text + "+" + comboBox6.Text + "+" + comboBox7.Text;
                Process.Start(test1);
            }
        }

        /// <summary>
        /// Handle Hot key events
        /// </summary>
        /// <param name="key"></param>
        private void HandleHotkey(String key)
        {
            switch (key)
            {
                case "1":
                    if (this.WindowState == FormWindowState.Minimized)
                    {
                        this.WindowState = FormWindowState.Normal;
                    }
                    this.Activate();
                    break;
                case "2":
                    this.WindowState = FormWindowState.Minimized;
                    break;
                case "3":
                    IEAccessible ie2 = new IEAccessible();
                    SHDocVw.ShellWindows shellWindows2 = new SHDocVw.ShellWindows();
                    foreach (SHDocVw.InternetExplorer window in shellWindows2)
                    {
                        ie2.Focus((IntPtr)window.HWND, getSite(0));
                    }
                    break;
                case "4":
                    IEAccessible ie1 = new IEAccessible();
                    SHDocVw.ShellWindows shellWindows1 = new SHDocVw.ShellWindows();
                    foreach (SHDocVw.InternetExplorer window in shellWindows1)
                    {
                        ie1.Focus((IntPtr)window.HWND, getSite(1));
                    }
                    break;
                case "5":
                    IEAccessible ie3 = new IEAccessible();
                    SHDocVw.ShellWindows shellWindows3 = new SHDocVw.ShellWindows();
                    foreach (SHDocVw.InternetExplorer window in shellWindows3)
                    {
                        ie3.Focus((IntPtr)window.HWND, getSite(2));
                    }
                    break;
                case "6":
                    // --
                    Process[] processes = Process.GetProcessesByName("iexplore");
                    foreach (var item in processes)
                    {
                        //ShowWindow(item.MainWindowHandle, SW_HIDE);
                        SetForegroundWindow(item.MainWindowHandle);
                    }
                    // --
                    break;
            }
        }

        /// <summary>
        /// Get Pressed Key
/// </summary>
/// <param name="LParam"></param>
/// <returns></returns>
private Keys GetKey(IntPtr LParam)
{
   return (Keys)((LParam.ToInt32()) >> 16);
}

/// <summary>
/// Handle Key Event
/// </summary>
/// <param name="m"></param>
protected override void WndProc(ref Message m)
{
   if (m.Msg == Constants.WM_HOTKEY_MSG_ID)
   {
      switch (GetKey(m.LParam))
      {
         case Keys.D1:
            // the hotkey key is (Shift + 1)
            HandleHotkey("1");
            break;
         case Keys.D2:
            // the hotkey key is (Shift + 2)
            HandleHotkey("2");
            break;
         case Keys.D3:
            // the hotkey key is (Shift + 3)
            HandleHotkey("6");
            break;
         case Keys.A:
            // the hotkey key is (Shift + a)
            HandleHotkey("3");
            break;
         case Keys.S:
            // the hotkey key is (Shift + i)
            HandleHotkey("4");
            break;
         case Keys.D:
            // the hotkey key is (Shift + k)
            HandleHotkey("5");
            break;
      }
   }

   base.WndProc(ref m);
}

/// <summary>
/// Form Load Event
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void Form1_Load(object sender, EventArgs e)
{
   ghk1.Register();
   ghk2.Register();
   ghk3.Register();
   ghk4.Register();
   ghk5.Register();
   ghk6.Register();
}
```

```
/// <summary>
/// Form Close Event
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
   ghk1.Unregiser();
   ghk2.Unregiser();
   ghk3.Unregiser();
   ghk4.Unregiser();
   ghk5.Unregiser();
   ghk6.Unregiser();
}

#endregion

#region NAudio
/// <summary>
/// Start recording from MIC
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button2_Click(object sender, EventArgs e)
{
   pictureBox2.BackColor = Color.Red;
   if (!debug)
   {
      // stop recognize before recode and filter
      speechRecognitionEngine.RecognizeAsyncCancel();
      speechRecognitionEngine.SetInputToNull();
   }

   // wait
   try
   {
      Thread.Sleep(10);
   }
   catch (Exception)
   {

   }

   waveSource = new WaveIn();
   waveSource.DataAvailable += new EventHandler<WaveInEventArgs>(waveIn_DataAvailable);
   waveSource.RecordingStopped += new EventHandler(waveSource_RecordingStopped);
   filter = BiQuadFilter.LowPassFilter(waveSource.WaveFormat.SampleRate, 10000, 1);
   waveFile = new WaveFileWriter(@"Test0001.wav", waveSource.WaveFormat);
   waveSource.StartRecording();
}

/// <summary>
/// Filter available data
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
void waveIn_DataAvailable(object sender, WaveInEventArgs e)
{
   short g = 0;
   var floatBuffer = new List<float>();
   for (int index = 0; index < e.BytesRecorded; index += 2)
   {
      short sample = BitConverter.ToInt16(e.Buffer, index);
```

```
        /**
         * Gain
         */
        float sample32 = (float)sample + g;
        sample32 /= (float)Int16.MaxValue;
        floatBuffer.Add(sample32);
    }

    /**
     * low pass filtering
     */
    for (int i = 0; i < floatBuffer.Count; i++)
    {
        waveFile.WriteSample(filter.Transform(floatBuffer[i]));
        waveFile.Flush();
    }
}

/// <summary>
/// Recode stop event
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
void waveSource_RecordingStopped(object sender, EventArgs e)
{

    if (waveSource != null)
    {
        waveSource.Dispose();
        waveSource = null;
    }

    if (waveFile != null)
    {
        waveFile.Dispose();
        waveFile = null;
    }

    if (!debug)
    {
        recognize();
    }
}

/// <summary>
/// Stop recording from MIC
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button3_Click(object sender, EventArgs e)
{
    pictureBox2.BackColor = Color.Green;
    if (waveSource != null)
    {
        waveSource.StopRecording();
    }
    else
    {
        if (!debug)
        {
            recognize();
        }
```

```
            }
        }

        /// <summary>
        /// Start recognize
        /// </summary>
        private void recognize()
        {
            speechRecognitionEngine.SetInputToWaveFile("Test0001.wav");
            speechRecognitionEngine.RecognizeAsync(RecognizeMode.Multiple);
        }

        #endregion

        #region clustering
        /// <summary>
        /// Init data for clustering
        /// </summary>
        private void Populatecluster()
        {
            Dictionary<String, Dictionary<String, String[]>> dic1 = new Dictionary<string, Dictionary<String,
String[]>>();
            try
            {
                string[] lines = File.ReadAllLines(Environment.CurrentDirectory + "\\Cluster.txt");
                foreach (string line in lines)
                {
                    // split the line
                    var parts = line.Split(new char[] { ':' });

                    if (dic1.ContainsKey(parts[0]))
                    {
                        Dictionary<String, String[]> tdic = dic1[parts[0]];
                        tdic[parts[1]] = parts[2].Split(new char[] { ',' });
                    }
                    else
                    {
                        Dictionary<String, String[]> tdic = new Dictionary<string, string[]>();
                        tdic[parts[1]] = parts[2].Split(new char[] { ',' });
                        dic1[parts[0]] = tdic;
                    }
                }
            }
            catch (Exception ex)
            {
                throw ex;
            }

            /// teach BayesClassifier
            foreach (KeyValuePair<String, Dictionary<String, String[]>> entry in dic1)
            {
                BayesClassifier.Classifier m_Classifier1 = new BayesClassifier.Classifier();
                vm_Classifier.Add(m_Classifier1);
                Teach(entry.Value, m_Classifier1);
            }

        }

        /// <summary>
        /// Training data
        /// </summary>
        /// <param name="tdata"></param>
        private void Teach(Dictionary<String, String[]> tdata)
```

```
{
   foreach (KeyValuePair<String, String[]> entry in tdata)
   {
      m_Classifier.TeachPhrases(entry.Key, entry.Value);
   }
}

/// <summary>
/// Training data
/// </summary>
/// <param name="tdata"></param>
/// <param name="m_Classifier1"></param>
private void Teach(Dictionary<String, String[]> tdata, BayesClassifier.Classifier m_Classifier1)
{
   foreach (KeyValuePair<String, String[]> entry in tdata)
   {
      m_Classifier1.TeachPhrases(entry.Key, entry.Value);
   }
}

/// <summary>
/// Classify into cluster
/// </summary>
/// <param name="data"></param>
/// <returns></returns>
private string Classify(String data)
{

   /*
    * Example data => "FTTH EMAIL"
    */

   // convert string to stream
   byte[] byteArray = Encoding.UTF8.GetBytes(data);
   MemoryStream stream = new MemoryStream(byteArray);
   // convert stream to string
   StreamReader reader = new StreamReader(stream);
   Dictionary<string, double> score = m_Classifier.Classify(reader);
   ArrayList myAL = new ArrayList();

   foreach (string c in score.Keys)
   {
      string key = c;
      double val = score[c];
      myAL.Add(val);
   }

   myAL.Sort();
   double max = (double)myAL[myAL.Count-1];
   ArrayList fi = new ArrayList();

   foreach (string c in score.Keys)
   {
      string key = c;
      double val = score[c];
      if (max == val)
         fi.Add(key);
   }

   return string.Join("|", fi.ToArray());
}

/// <summary>
```

```
        /// Classify into cluster
        /// </summary>
        /// <param name="data"></param>
        /// <param name="m_Classifier1"></param>
        /// <returns></returns>
        private string Classify(String data, BayesClassifier.Classifier m_Classifier1)
        {

            /*
             * Example data => "FTTH EMAIL"
             */

            // convert string to stream
            byte[] byteArray = Encoding.UTF8.GetBytes(data);
            MemoryStream stream = new MemoryStream(byteArray);
            // convert stream to string
            StreamReader reader = new StreamReader(stream);
            Dictionary<string, double> score = m_Classifier1.Classify(reader);
            ArrayList myAL = new ArrayList();

            foreach (string c in score.Keys)
            {
                string key = c;
                double val = score[c];
                myAL.Add(val);
            }

            myAL.Sort();
            double max = (double)myAL[myAL.Count - 1];
            ArrayList fi = new ArrayList();

            foreach (string c in score.Keys)
            {
                string key = c;
                double val = score[c];
                if (max == val)
                    fi.Add(key);
            }

            return string.Join("|", fi.ToArray());
        }

        #endregion

        #region spellChecker
        /// <summary>
        /// capture words like "test test"
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="args"></param>
        private void SpellChecker_DoubledWord(object sender, NetSpell.SpellChecker.SpellingEventArgs
args)
        {
            // update text
            string s = spellChecker.Text;
        }

        /// <summary>
        /// capture words like "test"
        /// this word is correct
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="args"></param>
        private void SpellChecker_EndOfText(object sender, System.EventArgs args)
        {
            // update text
            string s = spellChecker.Text;
            textBox1.Text = textBox1.Text + "," + s;
        }

        /// <summary>
        /// capture words like "tes"
        /// this word is incorrect
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="args"></param>
        private void SpellChecker_MisspelledWord(object sender, NetSpell.SpellChecker.SpellingEventArgs
args)
        {
            // update text
            string s = spellChecker.Text;
        }

        /// <summary>
        /// check spellings of the words
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void textBox1_TextChanged(object sender, EventArgs e)
        {
            string temp = textBox1.Text.Trim();
            string word = (temp.Contains(",")) ? temp.Split(',').Last() : temp;
            if (word != "")
            {
                spellChecker.Text = word;
            }

            if (!temp.Contains(","))
            {
                prevrecognized = temp;
            }
            else
            {
                prevrecognized = temp.Split(',')[temp.Split(',').Length - 1];
            }
        }


        /// <summary>
        /// check spellings
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button5_Click(object sender, EventArgs e)
        {
            spellChecker.SpellCheck();
        }

        #endregion

        #region Sites

        /// <summary>
        /// load sites for hot keys
        /// </summary>
        private void loadSites()
```

```
{
    try
    {
        string[] lines = File.ReadAllLines(Environment.CurrentDirectory + "\\SITES.txt");
        foreach (string line in lines)
        {
            sites.Add(line);
        }
    }
    catch (Exception ex)
    {
        throw ex;
    }
}

/// <summary>
/// get sites by index
/// </summary>
/// <param name="i"></param>
/// <returns></returns>
private string getSite(int i)
{
    try
    {
        string value = sites[i] as string;
        return value;
    }
    catch (Exception e)
    {
        return "not avail";
    }
}

#endregion

#region Markov

/// <summary>
/// Returns the number of steps required to transform the source string
/// into the target string.
/// </summary>
public static int ComputeLevenshteinDistance(string source, string target)
{
    if (string.IsNullOrEmpty(source))
        return string.IsNullOrEmpty(target) ? 0 : target.Length;

    if (string.IsNullOrEmpty(target))
        return string.IsNullOrEmpty(source) ? 0 : source.Length;

    int sourceLength = source.Length;
    int targetLength = target.Length;
    int[,] distance = new int[sourceLength + 1, targetLength + 1];

    // Step 1
    for (int i = 0; i <= sourceLength; distance[i, 0] = i++) ;
    for (int j = 0; j <= targetLength; distance[0, j] = j++) ;
    for (int i = 1; i <= sourceLength; i++)
    {
        for (int j = 1; j <= targetLength; j++)
        {
            // Step 2
            int cost = (target[j - 1] == source[i - 1]) ? 0 : 1;
            // Step 3
```

```
                distance[i, j] = Math.Min(Math.Min(distance[i - 1, j] + 1, distance[i, j - 1] + 1), distance[i - 1, j -
1] + cost);
            }
        }

        return distance[sourceLength, targetLength];
    }


    /// <summary>
    /// Calculate percentage similarity of two strings
    /// <param name="source">Source String to Compare with</param>
    /// <param name="target">Targeted String to Compare</param>
    /// <returns>Return Similarity between two strings from 0 to 1.0</returns>
    /// </summary>
    public static double CalculateSimilarity(string source, string target)
    {
        if (string.IsNullOrEmpty(source))
            return string.IsNullOrEmpty(target) ? 1 : 0;


        if (string.IsNullOrEmpty(target))
            return string.IsNullOrEmpty(source) ? 1 : 0;


        double stepsToSame = ComputeLevenshteinDistance(source, target);
        return (1.0 - (stepsToSame / (double)Math.Max(source.Length, target.Length)));
    }


    /// <summary>
    /// tokenise a string into words (regex definition of word)
    /// </summary>
    /// <param name="subject"></param>
    /// <returns></returns>
    private static IEnumerable<string> Split(string subject)
    {
        List<string> tokens = new List<string>();
        Regex regex = new Regex(@"(\W+)");
        string[] arr = regex.Split(subject);
        for (int i = 0; i < arr.Length; i++)
        {
            if (arr[i].Equals(" "))
                continue;

            if (arr[i].Equals(""))
                continue;

            if (arr[i].Contains("\n"))
                continue;

            if (arr[i].Contains("."))
                continue;

            tokens.Add(arr[i]);
        }

        return tokens;
    }


    /// <summary>
    /// Preprocess string
```

```
        /// </summary>
        /// <param name="p"></param>
        /// <returns></returns>
        private static string Tidy(string p)
        {
            string result = p.Replace('\t', ' ');
            string compress = result;

            do
            {
                result = compress;
                compress = result.Replace(" ", " ");
            }
            while (result != compress);
            return result;
        }

        /// <summary>
        /// load Markov
        /// </summary>
        private void LoadMarkov()
        {
            try
            {
                string seed = "";

                string[] lines = File.ReadAllLines(Environment.CurrentDirectory + "\\Markov.txt");
                foreach (string line in lines)
                {
                    seed = line;
                    break;
                }

                seed = Tidy(seed);

                // tokenise the input string
                var seedList = new List<string>(Split(seed.ToLower()));
                // create a chain with a window size of 4
                chain = new Chain<string>(seedList, 4);

                string[] startw = File.ReadAllLines(Environment.CurrentDirectory + "\\MarkovStartW.txt");

                for (int i = 0; i < startw.Length; i++)
                {
                    chainstart.Add(startw[i]);
                }

            }
            catch (Exception ex)
            {
                throw ex;
            }
        }

        #endregion

    }
}
```

Appendix B

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

namespace MSC
{
    /// <summary>
    /// BiQuad filter
    /// </summary>
    public class BiQuadFilter
    {
        // coefficients
        private double a0;
        private double a1;
        private double a2;
        private double a3;
        private double a4;

        // state
        private float x1;
        private float x2;
        private float y1;
        private float y2;

        /// <summary>
        /// Passes a single sample through the filter
        /// </summary>
        /// <param name="inSample">Input sample</param>
        /// <returns>Output sample</returns>
        public float Transform(float inSample)
        {
            // compute result
            var result = a0 * inSample + a1 * x1 + a2 * x2 - a3 * y1 - a4 * y2;

            // shift x1 to x2, sample to x1
            x2 = x1;
            x1 = inSample;

            // shift y1 to y2, result to y1
            y2 = y1;
            y1 = (float)result;

            return y1;
            //return inSample;
        }

        private void SetCoefficients(double aa0, double aa1, double aa2, double b0, double b1, double b2)
        {
            // precompute the coefficients
            a0 = b0 / aa0;
            a1 = b1 / aa0;
            a2 = b2 / aa0;
            a3 = aa1 / aa0;
            a4 = aa2 / aa0;
        }

        /// <summary>
        /// Set this up as a low pass filter
        /// </summary>
/// <param name="sampleRate">Sample Rate</param>
/// <param name="cutoffFrequency">Cut-off Frequency</param>
/// <param name="q">Bandwidth</param>
public void SetLowPassFilter(float sampleRate, float cutoffFrequency, float q)
{
    // H(s) = 1 / (s^2 + s/Q + 1)
    var w0 = 2 * Math.PI * cutoffFrequency / sampleRate;
    var cosw0 = Math.Cos(w0);
    var alpha = Math.Sin(w0) / (2 * q);

    var b0 = (1 - cosw0) / 2;
    var b1 = 1 - cosw0;
    var b2 = (1 - cosw0) / 2;
    var aa0 = 1 + alpha;
    var aa1 = -2 * cosw0;
    var aa2 = 1 - alpha;
    SetCoefficients(aa0, aa1, aa2, b0, b1, b2);
}

/// <summary>
/// Set this up as a peaking EQ
/// </summary>
/// <param name="sampleRate">Sample Rate</param>
/// <param name="centreFrequency">Centre Frequency</param>
/// <param name="q">Bandwidth (Q)</param>
/// <param name="dbGain">Gain in decibels</param>
public void SetPeakingEq(float sampleRate, float centreFrequency, float q, float dbGain)
{
    // H(s) = (s^2 + s*(A/Q) + 1) / (s^2 + s/(A*Q) + 1)
    var w0 = 2 * Math.PI * centreFrequency / sampleRate;
    var cosw0 = Math.Cos(w0);
    var sinw0 = Math.Sin(w0);
    var alpha = sinw0 / (2 * q);
    var a = Math.Pow(10, dbGain / 40);     // TODO: should we square root this value?

    var b0 = 1 + alpha * a;
    var b1 = -2 * cosw0;
    var b2 = 1 - alpha * a;
    var aa0 = 1 + alpha / a;
    var aa1 = -2 * cosw0;
    var aa2 = 1 - alpha / a;
    SetCoefficients(aa0, aa1, aa2, b0, b1, b2);
}

/// <summary>
/// Set this as a high pass filter
/// </summary>
public void SetHighPassFilter(float sampleRate, float cutoffFrequency, float q)
{
    // H(s) = s^2 / (s^2 + s/Q + 1)
    var w0 = 2 * Math.PI * cutoffFrequency / sampleRate;
    var cosw0 = Math.Cos(w0);
    var alpha = Math.Sin(w0) / (2 * q);

    var b0 = (1 + cosw0) / 2;
    var b1 = -(1 + cosw0);
    var b2 = (1 + cosw0) / 2;
    var aa0 = 1 + alpha;
    var aa1 = -2 * cosw0;
    var aa2 = 1 - alpha;
    SetCoefficients(aa0, aa1, aa2, b0, b1, b2);
}
```

```
        /// <summary>
        /// Create a low pass filter
        /// </summary>
        public static BiQuadFilter LowPassFilter(float sampleRate, float cutoffFrequency, float q)
        {
            var filter = new BiQuadFilter();
            filter.SetLowPassFilter(sampleRate, cutoffFrequency, q);
            return filter;
        }

        /// <summary>
        /// Create a High pass filter
        /// </summary>
        public static BiQuadFilter HighPassFilter(float sampleRate, float cutoffFrequency, float q)
        {
            var filter = new BiQuadFilter();
            filter.SetHighPassFilter(sampleRate, cutoffFrequency, q);
            return filter;
        }

        /// <summary>
        /// Create a bandpass filter with constant skirt gain
        /// </summary>
        public static BiQuadFilter BandPassFilterConstantSkirtGain(float sampleRate, float centreFrequency,
float q)
        {
            // H(s) = s / (s^2 + s/Q + 1)  (constant skirt gain, peak gain = Q)
            var w0 = 2 * Math.PI * centreFrequency / sampleRate;
            var cosw0 = Math.Cos(w0);
            var sinw0 = Math.Sin(w0);
            var alpha = sinw0 / (2 * q);

            var b0 = sinw0 / 2; // =   Q*alpha
            var b1 = 0;
            var b2 = -sinw0 / 2; // =  -Q*alpha
            var a0 = 1 + alpha;
            var a1 = -2 * cosw0;
            var a2 = 1 - alpha;
            return new BiQuadFilter(a0, a1, a2, b0, b1, b2);
        }

        /// <summary>
        /// Create a bandpass filter with constant peak gain
        /// </summary>
        public static BiQuadFilter BandPassFilterConstantPeakGain(float sampleRate, float centreFrequency,
float q)
        {
            // H(s) = (s/Q) / (s^2 + s/Q + 1)      (constant 0 dB peak gain)
            var w0 = 2 * Math.PI * centreFrequency / sampleRate;
            var cosw0 = Math.Cos(w0);
            var sinw0 = Math.Sin(w0);
            var alpha = sinw0 / (2 * q);

            var b0 = alpha;
            var b1 = 0;
            var b2 = -alpha;
            var a0 = 1 + alpha;
            var a1 = -2 * cosw0;
            var a2 = 1 - alpha;
            return new BiQuadFilter(a0, a1, a2, b0, b1, b2);
        }

        /// <summary>
        /// Creates a notch filter
        /// </summary>
        public static BiQuadFilter NotchFilter(float sampleRate, float centreFrequency, float q)
        {
            // H(s) = (s^2 + 1) / (s^2 + s/Q + 1)
            var w0 = 2 * Math.PI * centreFrequency / sampleRate;
            var cosw0 = Math.Cos(w0);
            var sinw0 = Math.Sin(w0);
            var alpha = sinw0 / (2 * q);

            var b0 = 1;
            var b1 = -2 * cosw0;
            var b2 = 1;
            var a0 = 1 + alpha;
            var a1 = -2 * cosw0;
            var a2 = 1 - alpha;
            return new BiQuadFilter(a0, a1, a2, b0, b1, b2);
        }

        /// <summary>
        /// Creaes an all pass filter
        /// </summary>
        public static BiQuadFilter AllPassFilter(float sampleRate, float centreFrequency, float q)
        {
            //H(s) = (s^2 - s/Q + 1) / (s^2 + s/Q + 1)
            var w0 = 2 * Math.PI * centreFrequency / sampleRate;
            var cosw0 = Math.Cos(w0);
            var sinw0 = Math.Sin(w0);
            var alpha = sinw0 / (2 * q);

            var b0 = 1 - alpha;
            var b1 = -2 * cosw0;
            var b2 = 1 + alpha;
            var a0 = 1 + alpha;
            var a1 = -2 * cosw0;
            var a2 = 1 - alpha;
            return new BiQuadFilter(a0, a1, a2, b0, b1, b2);
        }

        /// <summary>
        /// Create a Peaking EQ
        /// </summary>
        public static BiQuadFilter PeakingEQ(float sampleRate, float centreFrequency, float q, float dbGain)
        {
            var filter = new BiQuadFilter();
            filter.SetPeakingEq(sampleRate, centreFrequency, q, dbGain);
            return filter;
        }

        /// <summary>
        /// H(s) = A * (s^2 + (sqrt(A)/Q)*s + A)/(A*s^2 + (sqrt(A)/Q)*s + 1)
        /// </summary>
        /// <param name="sampleRate"></param>
        /// <param name="cutoffFrequency"></param>
        /// <param name="shelfSlope">a "shelf slope" parameter (for shelving EQ only).
        /// When S = 1, the shelf slope is as steep as it can be and remain monotonically
        /// increasing or decreasing gain with frequency.  The shelf slope, in dB/octave,
        /// remains proportional to S for all other values for a fixed f0/Fs and dBgain.</param>
        /// <param name="dbGain">Gain in decibels</param>
        public static BiQuadFilter LowShelf(float sampleRate, float cutoffFrequency, float shelfSlope, float
dbGain)
        {
            var w0 = 2 * Math.PI * cutoffFrequency / sampleRate;
            var cosw0 = Math.Cos(w0);
            var sinw0 = Math.Sin(w0);
            var a = Math.Pow(10, dbGain / 40);     // TODO: should we square root this value?
            var alpha = sinw0 / 2 * Math.Sqrt((a + 1 / a) * (1 / shelfSlope - 1) + 2);
            var temp = 2 * Math.Sqrt(a) * alpha;

            var b0 = a * ((a + 1) - (a - 1) * cosw0 + temp);
            var b1 = 2 * a * ((a - 1) - (a + 1) * cosw0);
            var b2 = a * ((a + 1) - (a - 1) * cosw0 - temp);
            var a0 = (a + 1) + (a - 1) * cosw0 + temp;
            var a1 = -2 * ((a - 1) + (a + 1) * cosw0);
            var a2 = (a + 1) + (a - 1) * cosw0 - temp;
            return new BiQuadFilter(a0, a1, a2, b0, b1, b2);
        }

        /// <summary>
        /// H(s) = A * (A*s^2 + (sqrt(A)/Q)*s + 1)/(s^2 + (sqrt(A)/Q)*s + A)
        /// </summary>
        /// <param name="sampleRate"></param>
        /// <param name="cutoffFrequency"></param>
        /// <param name="shelfSlope"></param>
        /// <param name="dbGain"></param>
        /// <returns></returns>
        public static BiQuadFilter HighShelf(float sampleRate, float cutoffFrequency, float shelfSlope, float
dbGain)
        {
            var w0 = 2 * Math.PI * cutoffFrequency / sampleRate;
            var cosw0 = Math.Cos(w0);
            var sinw0 = Math.Sin(w0);
            var a = Math.Pow(10, dbGain / 40);     // TODO: should we square root this value?
            var alpha = sinw0 / 2 * Math.Sqrt((a + 1 / a) * (1 / shelfSlope - 1) + 2);
            var temp = 2 * Math.Sqrt(a) * alpha;

            var b0 = a * ((a + 1) + (a - 1) * cosw0 + temp);
            var b1 = -2 * a * ((a - 1) + (a + 1) * cosw0);
            var b2 = a * ((a + 1) + (a - 1) * cosw0 - temp);
            var a0 = (a + 1) - (a - 1) * cosw0 + temp;
            var a1 = 2 * ((a - 1) - (a + 1) * cosw0);
            var a2 = (a + 1) - (a - 1) * cosw0 - temp;
            return new BiQuadFilter(a0, a1, a2, b0, b1, b2);
        }

        private BiQuadFilter()
        {
            // zero initial samples
            x1 = x2 = 0;
            y1 = y2 = 0;
        }

        private BiQuadFilter(double a0, double a1, double a2, double b0, double b1, double b2)
        {
            SetCoefficients(a0, a1, a2, b0, b1, b2);

            // zero initial samples
            x1 = x2 = 0;
            y1 = y2 = 0;
        }
    }
}
```

Appendix C

```
using System;
using System.Collections.Generic;
using System.Text;

namespace BayesClassifier
{
	/// <summary>
	/// Naive Bayesian classifier</summary>
	/// <remarks>
	/// It suppports exclusion of words but not Phrases
	/// </remarks>
	public class Classifier : BayesClassifier.IClassifier
	{
		SortedDictionary<string, ICategory> m_Categories;
		ExcludedWords m_ExcludedWords;

		public Classifier()
		{
			m_Categories = new SortedDictionary<string, ICategory>();
			m_ExcludedWords = new ExcludedWords();
			m_ExcludedWords.InitDefault();
		}

		/// <summary>
		/// Gets total number of word occurences over all categories</summary>
		int CountTotalWordsInCategories()
		{
			int total = 0;
			foreach (Category cat in m_Categories.Values)
			{
				total += cat.TotalWords;
			}
			return total;
		}

		/// <summary>
		/// Gets or creates a category</summary>
		ICategory GetOrCreateCategory(string cat)
		{
			ICategory c;
			if (!m_Categories.TryGetValue(cat, out c))
			{
				c = new Category(cat, m_ExcludedWords);
				m_Categories.Add(cat, c);
			}
			return c;
		}

		/// <summary>
		/// Trains this Category from a word or phrase<\summary>
		public void TeachPhrases(string cat, string[] phrases)
		{
			GetOrCreateCategory(cat).TeachPhrases(phrases);
		}

		/// <summary>
		/// Trains this Category from a word or phrase<\summary>
		public void TeachCategory(string cat, System.IO.TextReader tr)
		{
			GetOrCreateCategory(cat).TeachCategory(tr);
		}

		/// <summary>
		/// Classifies a text<\summary>
		/// <returns>
		/// returns classification values for the text, the higher, the better is the match.</returns>
		public Dictionary<string, double> Classify(System.IO.StreamReader tr)
		{
			Dictionary<string, double> score = new Dictionary<string, double>();
			foreach (KeyValuePair<string, ICategory> cat in m_Categories)
			{
				score.Add(cat.Value.Name, 0.0);
			}

			EnumerableCategory words_in_file = new EnumerableCategory("",
m_ExcludedWords);
			words_in_file.TeachCategory(tr);

			foreach (KeyValuePair<string, PhraseCount> kvp1 in words_in_file)
			{
				PhraseCount pc_in_file = kvp1.Value;
				foreach (KeyValuePair<string, ICategory> kvp in m_Categories)
				{
				ICategory cat = kvp.Value;
				int count = cat.GetPhraseCount(pc_in_file.RawPhrase);
				if (0 < count)
				{
				score[cat.Name] += System.Math.Log((double)count /
(double)cat.TotalWords);
				}
				else
				{
				score[cat.Name] += System.Math.Log(0.01 / (double)cat.TotalWords);
				}
				System.Diagnostics.Trace.WriteLine(pc_in_file.RawPhrase.ToString()
+ "(" +
						cat.Name + ")" + score[cat.Name]);
				}
			}
			foreach (KeyValuePair<string, ICategory> kvp in m_Categories)
			{
			ICategory cat = kvp.Value;
				score[cat.Name] += System.Math.Log((double)cat.TotalWords /
			(double)this.CountTotalWordsInCategories());
			}
			return score;
		}
	}
}
```

```
using System;
using System.Collections.Generic;

namespace BayesClassifier
{
   #region Interfaces
            /// <summary>
            /// Classifier methods.
            /// </summary>
            public interface IClassifier
            {
                        /// <summary>
                        /// Trains this Category from a word or phrase<\summary>
                        void TeachPhrases(string cat, string[] phrases);

                        /// <summary>
                        /// Trains this Category from a word or phrase<\summary>
                        void TeachCategory(string cat, System.IO.TextReader tr);

                        /// <summary>
                        /// Classifies a text<\summary>
                        /// <returns>
                        /// returns classification values for the text, the higher, the better is the match.</returns>
                        Dictionary<string, double> Classify(System.IO.StreamReader tr);
            }

            /// <summary>
            /// Category methods.
            /// </summary>
            interface ICategory
            {
                        string Name { get; }
                        /// <summary>
                        /// Reset all trained data<\summary>
                        void Reset();
                        /// <summary>
                        /// Gets a PhraseCount for Phrase or 0 if phrase not present<\summary>
                        int GetPhraseCount(string phrase);

                        /// <summary>
                        /// Trains this Category from a file<\summary>
                        void TeachCategory(System.IO.TextReader reader);
                        /// <summary>
                        /// Trains this Category from a word or phrase<\summary>
                        void TeachPhrase(string rawPhrase);
                        /// <summary>
                        /// Trains this Category from a word or phrase array<\summary>
                        void TeachPhrases(string[] words);
                        /// <value>
                        /// Gets total number of word occurences in this category</value>
                        int TotalWords { get; }
            }
   #endregion
}
```

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Text.RegularExpressions;

namespace BayesClassifier
{
	/// <summary>
	/// stores occurence counter for words or phrases</summary>
	class PhraseCount
	{
		string m_RawPhrase;

		public PhraseCount(string rawPhrase)
		{
			m_RawPhrase = rawPhrase;
			m_Count = 0;
		}

		/// <value>
		/// Stores the raw Phrase, the real matching phrase is stored as key to this element</value>
		/// <seealso cref="DePhrase(string)">
		/// See DePhrase </seealso>
		public string RawPhrase
		{
			get { return m_RawPhrase; }
			//set { m_RawPhrase = value; }
		}

		int m_Count;

		/// <value>
		/// Count Accessor</value>
		public int Count
		{
			get { return m_Count; }
			set { m_Count = value; }
		}
	}
}
```

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Text;
using System.Text.RegularExpressions;
namespace BayesClassifier
{
	/// <summary>
	/// Represents a Enumerable Bayesian category - that is contains a list of phrases with their
occurence counts <\summary>
	class EnumerableCategory : Category, IEnumerable<KeyValuePair<string, PhraseCount>>
	{
		public EnumerableCategory(string Cat, ExcludedWords Excluded) : base(Cat, Excluded)
		{
		}

		IEnumerator IEnumerable.GetEnumerator()
		{
			return GetEnumerator();
		}
		public IEnumerator<KeyValuePair<string, PhraseCount>> GetEnumerator()
		{
			return m_Phrases.GetEnumerator();
		}

	}


	/// <summary>
	/// Represents a Bayesian category - that is contains a list of phrases with their occurence counts
<\summary>
	class Category : BayesClassifier.ICategory
	{
		protected System.Collections.Generic.SortedDictionary<string, PhraseCount> m_Phrases;
		int m_TotalWords;
		string m_Name;
		ExcludedWords m_Excluded;

		public string Name
		{
			get { return m_Name; }
			//set { m_Name = value; }
		}
		/// <value>
		/// Gets total number of word occurences in this category</value>
		public int TotalWords
		{
			get { return m_TotalWords; }
			//set { m_TotalWords = value; }
		}

		public Category(string cat, ExcludedWords excluded)
		{
			m_Phrases = new SortedDictionary<string, PhraseCount>();
			m_Excluded = excluded;
			m_Name = cat;
		}

		/// <summary>
		/// Gets a Count for Phrase or 0 if not present<\summary>
		public int GetPhraseCount(string phrase)
		{
			PhraseCount pc;
		if (m_Phrases.TryGetValue(phrase, out pc))
			return pc.Count;
		else
			return 0;
	}

	/// <summary>
	/// Reset all trained data<\summary>
	public void Reset()
	{
		m_TotalWords = 0;
		m_Phrases.Clear();
	}

	System.Collections.Generic.SortedDictionary<string, PhraseCount> Phrases
	{
		get { return m_Phrases; }
		//set { m_Phrases = value; }
	}

	/// <summary>
	/// Trains this Category from a file<\summary>
	public void TeachCategory(System.IO.TextReader reader)
	{
		//System.Diagnostics.Debug.Assert(line.Length < 512);
		Regex re = new Regex(@"(\w+)\W*", RegexOptions.Compiled);
		string line;
		while (null != (line = reader.ReadLine()))
		{
			Match m = re.Match(line);
			while (m.Success)
			{
				string word = m.Groups[1].Value;
				TeachPhrase(word);
				m = m.NextMatch();
			}
		}
	}

	/// <summary>
	/// Trains this Category from a word or phrase array<\summary>
	/// <seealso cref="DePhrase(string)">
	/// See DePhrase </seealso>
	public void TeachPhrases(string[] words)
	{
		foreach (string word in words)
		{
			TeachPhrase(word);
		}
	}

	/// <summary>
	/// Trains this Category from a word or phrase<\summary>
	/// <seealso cref="DePhrase(string)">
	/// See DePhrase </seealso>
	public void TeachPhrase(string rawPhrase)
	{
		if ((null != m_Excluded) && (m_Excluded.IsExcluded(rawPhrase)))
			return;

		PhraseCount pc;
		string Phrase = DePhrase(rawPhrase);
		if (!m_Phrases.TryGetValue(Phrase, out pc))
```

```
                        {
                                    pc = new PhraseCount(rawPhrase);
                                    m_Phrases.Add(Phrase, pc);
                        }
                        pc.Count++;
                        m_TotalWords++;
            }

            static Regex ms_PhraseRegEx = new Regex(@"\W", RegexOptions.Compiled);

            /// <summary>
            /// Checks if a string is a phrase (that is a string with whitespaces)<\summary>
            /// <returns>
            /// true or false</returns>
            /// <seealso cref="DePhrase(string)">
            /// See DePhrase </seealso>
            public static bool CheckIsPhrase(string s)
            {
                        return ms_PhraseRegEx.IsMatch(s);
            }

            /// <summary>
            /// Trnasforms a string into a phrase (that is a string with whitespaces)<\summary>
            /// <returns>
            /// dephrased string</returns>
            /// <remarks>
            /// if something like "lone Rhino" is considered a sinlge Phrase, then our word matching
algorithm is
            /// is transforming it into a single Word "lone Rhino" -> "loneRhino"
            /// Currently this feature is not used!
            /// </remarks>
            public static string DePhrase(string s)
            {
                        return ms_PhraseRegEx.Replace(s, @"");
            }

      }
}
```

Appendix D

```
using Accessibility;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Runtime.InteropServices;
using System.Windows.Forms;

namespace ADDIProtocol
{
   public class IEAccessible
   {
      private enum OBJID : uint
      {
         OBJID_WINDOW = 0x00000000,
      }

      private const int IE_ACTIVE_TAB = 2097154;
      private const int CHILDID_SELF = 0;
      private IAccessible accessible;

      private IEAccessible[] Children
      {
         get
         {
            int num = 0;
            object[] res = GetAccessibleChildren(accessible, out num);

            if (res == null) return new IEAccessible[0];

            List<IEAccessible> list = new List<IEAccessible>(res.Length);

            foreach (object obj in res)
            {
               IAccessible acc = obj as IAccessible;

               if (acc != null) list.Add(new IEAccessible(acc));
            }

            return list.ToArray();
         }
      }

      private string Name
      {
         get
         {
            return accessible.get_accName(CHILDID_SELF);
         }
      }

      private int ChildCount
      {
         get
         {
            return accessible.accChildCount;
         }
      }

      public IEAccessible()
      { }
```

```
        public IEAccessible(IntPtr ieHandle, string tabCaptionToActivate)
        {
            int tabCount = GetTabCount(ieHandle);
            if (tabCount > 0)
            {
                AccessibleObjectFromWindow(GetDirectUIHWND(ieHandle), OBJID.OBJID_WINDOW, ref
accessible);

                if (accessible == null) throw new Exception();

                IEAccessible ieDirectUIHWND = new IEAccessible(ieHandle);

                foreach (IEAccessible accessor in ieDirectUIHWND.Children)
                {
                    foreach (IEAccessible child in accessor.Children)
                    {
                        foreach (IEAccessible tab in child.Children)
                        {
                            if (tab.Name.ToLower().Contains(tabCaptionToActivate.ToLower()))
                            {
                                tab.Activate();
                                return;
                            }
                        }
                    }
                }
            }
        }

        public bool Focus(IntPtr ieHandle, string tabCaptionToActivate)
        {

            int tabCount = GetTabCount(ieHandle);
            if (tabCount > 0)
            {
                AccessibleObjectFromWindow(GetDirectUIHWND(ieHandle), OBJID.OBJID_WINDOW, ref
accessible);
                if (accessible == null) throw new Exception();

                IEAccessible ieDirectUIHWND = new IEAccessible(ieHandle);
                foreach (IEAccessible accessor in ieDirectUIHWND.Children)
                {
                    foreach (IEAccessible child in accessor.Children)
                    {
                        foreach (IEAccessible tab in child.Children)
                        {
                            if (tab.Name.ToLower().Contains(tabCaptionToActivate.ToLower()))
                            {
                                tab.Activate();
                                return true;
                            }
                        }
                    }
                }
            }

            return false;
        }

        public bool Is(IntPtr ieHandle, string tabCaptionToActivate)
        {
            int tabCount = GetTabCount(ieHandle);
            if (tabCount > 0)
            {
                AccessibleObjectFromWindow(GetDirectUIHWND(ieHandle), OBJID.OBJID_WINDOW, ref
accessible);

                if (accessible == null) throw new Exception();

                IEAccessible ieDirectUIHWND = new IEAccessible(ieHandle);

                foreach (IEAccessible accessor in ieDirectUIHWND.Children)
                {
                    foreach (IEAccessible child in accessor.Children)
                    {
                        foreach (IEAccessible tab in child.Children)
                        {
                            if (tab.Name.ToLower().Contains(tabCaptionToActivate.ToLower()))
                            {
                                return true;
                            }
                        }
                    }
                }
            }

            return false;
        }

        public IEAccessible(IntPtr ieHandle, int tabIndexToActivate)
        {
            AccessibleObjectFromWindow(GetDirectUIHWND(ieHandle), OBJID.OBJID_WINDOW, ref
accessible);

            if (accessible == null) throw new Exception();

            int index = 0;
            IEAccessible ieDirectUIHWND = new IEAccessible(ieHandle);

            foreach (IEAccessible accessor in ieDirectUIHWND.Children)
            {
                foreach (IEAccessible child in accessor.Children)
                {
                    foreach (IEAccessible tab in child.Children)
                    {
                        if (tabIndexToActivate >= child.ChildCount - 1) return;

                        if (index == tabIndexToActivate)
                        {
                            tab.Activate();

                            return;
                        }

                        index++;
                    }
                }
            }
        }

        private IEAccessible(IntPtr ieHandle)
        {
            AccessibleObjectFromWindow(GetDirectUIHWND(ieHandle), OBJID.OBJID_WINDOW, ref
accessible);
```

```
            if (accessible == null) throw new Exception();
        }

        public string GetActiveTabUrl(IntPtr ieHandle)
        {
            AccessibleObjectFromWindow(GetDirectUIHWND(ieHandle), OBJID.OBJID_WINDOW, ref
accessible);

            if (accessible == null) throw new Exception();

            IEAccessible ieDirectUIHWND = new IEAccessible(ieHandle);

            foreach (IEAccessible accessor in ieDirectUIHWND.Children)
            {
                foreach (IEAccessible child in accessor.Children)
                {
                    foreach (IEAccessible tab in child.Children)
                    {
                        object tabIndex = tab.accessible.get_accState(CHILDID_SELF);

                        if ((int)tabIndex == IE_ACTIVE_TAB)
                        {
                            string description = tab.accessible.get_accDescription(CHILDID_SELF);

                            if (!string.IsNullOrEmpty(description))
                            {
                                if (description.Contains(Environment.NewLine))
                                {
                                    string url =
description.Substring(description.IndexOf(Environment.NewLine)).Trim();

                                    return url;
                                }
                            }
                        }
                    }
                }
            }

            return String.Empty;
        }

        public int GetActiveTabIndex(IntPtr ieHandle)
        {
            AccessibleObjectFromWindow(GetDirectUIHWND(ieHandle), OBJID.OBJID_WINDOW, ref
accessible);

            if (accessible == null) throw new Exception();

            int index = 0;
            IEAccessible ieDirectUIHWND = new IEAccessible(ieHandle);

            foreach (IEAccessible accessor in ieDirectUIHWND.Children)
            {
                foreach (IEAccessible child in accessor.Children)
                {
                    foreach (IEAccessible tab in child.Children)
                    {
                        object tabIndex = tab.accessible.get_accState(0);

                        if ((int)tabIndex == IE_ACTIVE_TAB) return index;

                        index++;
                    }
                }
            }

            return -1;
        }

        public string GetActiveTabCaption(IntPtr ieHandle)
        {
            AccessibleObjectFromWindow(GetDirectUIHWND(ieHandle), OBJID.OBJID_WINDOW, ref
accessible);

            if (accessible == null) throw new Exception();

            IEAccessible ieDirectUIHWND = new IEAccessible(ieHandle);

            foreach (IEAccessible accessor in ieDirectUIHWND.Children)
            {
                foreach (IEAccessible child in accessor.Children)
                {
                    foreach (IEAccessible tab in child.Children)
                    {
                        object tabIndex = tab.accessible.get_accState(0);


                        if ((int)tabIndex == IE_ACTIVE_TAB) return tab.Name;
                    }
                }
            }

            return String.Empty;
        }

        public List<string> GetTabCaptions(IntPtr ieHandle)
        {
            AccessibleObjectFromWindow(GetDirectUIHWND(ieHandle), OBJID.OBJID_WINDOW, ref
accessible);

            if (accessible == null) throw new Exception();

            IEAccessible ieDirectUIHWND = new IEAccessible(ieHandle);
            List<string> captionList = new List<string>();

            foreach (IEAccessible accessor in ieDirectUIHWND.Children)
            {
                foreach (IEAccessible child in accessor.Children)
                    foreach (IEAccessible tab in child.Children) captionList.Add(tab.Name);
            }

            if (captionList.Count > 0) captionList.RemoveAt(captionList.Count - 1);

            return captionList;
        }

        public int GetTabCount(IntPtr ieHandle)
        {
            AccessibleObjectFromWindow(GetDirectUIHWND(ieHandle), OBJID.OBJID_WINDOW, ref
accessible);

            if (accessible == null) return 0; // throw new Exception();

            IEAccessible ieDirectUIHWND = new IEAccessible(ieHandle);
            foreach (IEAccessible accessor in ieDirectUIHWND.Children)
            {
                foreach (IEAccessible child in accessor.Children)
                {
                    foreach (IEAccessible tab in child.Children) return child.ChildCount - 1;
                }
            }

            return 0;
        }

        //private IntPtr GetDirectUIHWND(IntPtr ieFrame)
        //{
        //    IntPtr directUI = FindWindowEx(ieFrame, IntPtr.Zero, "CommandBarClass", null);
        //    directUI = FindWindowEx(directUI, IntPtr.Zero, "ReBarWindow32", null);
        //    directUI = FindWindowEx(directUI, IntPtr.Zero, "TabBandClass", null);
        //    directUI = FindWindowEx(directUI, IntPtr.Zero, "DirectUIHWND", null);

        //    return directUI;
        //}

        private IntPtr GetDirectUIHWND(IntPtr ieFrame)
        {
            // For IE 8:
            var directUI = FindWindowEx(ieFrame, IntPtr.Zero, "CommandBarClass", null);
            directUI = FindWindowEx(directUI, IntPtr.Zero, "ReBarWindow32", null);
            directUI = FindWindowEx(directUI, IntPtr.Zero, "TabBandClass", null);
            directUI = FindWindowEx(directUI, IntPtr.Zero, "DirectUIHWND", null);

            if (directUI == IntPtr.Zero)
            {
                // For IE 9:
                directUI = FindWindowEx(ieFrame, IntPtr.Zero, "WorkerW", null);
                directUI = FindWindowEx(directUI, IntPtr.Zero, "ReBarWindow32", null);
                directUI = FindWindowEx(directUI, IntPtr.Zero, "TabBandClass", null);
                directUI = FindWindowEx(directUI, IntPtr.Zero, "DirectUIHWND", null);
            }

            return directUI;
        }

        private IEAccessible(IAccessible acc)
        {
            if (acc == null) throw new Exception();

            accessible = acc;
        }

        private void Activate()
        {
            accessible.accDoDefaultAction(CHILDID_SELF);
        }

        private static object[] GetAccessibleChildren(IAccessible ao, out int childs)
        {
            childs = 0;
            object[] ret = null;
            int count = ao.accChildCount;

            if (count > 0)
            {
                ret = new object[count];
                AccessibleChildren(ao, 0, count, ret, out childs);
```

```
            }

            return ret;
        }

        #region Interop
        [DllImport("user32.dll", SetLastError = true)]
        private static extern IntPtr FindWindowEx(IntPtr hwndParent, IntPtr hwndChildAfter, string lpszClass,
string lpszWindow);

        private static int AccessibleObjectFromWindow(IntPtr hwnd, OBJID idObject, ref IAccessible acc)
        {
            object obj = null;
            Guid guid = new Guid("{618736e0-3c3d-11cf-810c-00aa00389b71}"); // IAccessibleobject obj =
null;
            int num = AccessibleObjectFromWindow(hwnd, (uint)idObject, ref guid, ref obj);

            acc = (IAccessible)obj;

            return num;
        }

        [DllImport("oleacc.dll")]
        private static extern int AccessibleObjectFromWindow(IntPtr hwnd, uint id, ref Guid iid, [In, Out,
MarshalAs(UnmanagedType.IUnknown)] ref object ppvObject);

        [DllImport("oleacc.dll")]
        private static extern int AccessibleChildren(IAccessible paccContainer, int iChildStart, int cChildren,
[In, Out, MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 2)] object[] rgvarChildren, out int
pcObtained);
        #endregion
    }}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Runtime.InteropServices;

namespace MSC
{
    class GlobalHotkey
    {
        [DllImport("user32.dll")]
        private static extern bool RegisterHotKey(IntPtr hWnd, int id, int fsModifiers, int vk);

        [DllImport("user32.dll")]
        private static extern bool UnregisterHotKey(IntPtr hWnd, int id);

        private int modifier;
        private int key;
        private IntPtr hWnd;
        private int id;

        public GlobalHotkey(int modifier, Keys key, Form form)
        {
            this.modifier = modifier;
            this.key = (int)key;
            this.hWnd = form.Handle;
            id = this.GetHashCode();
        }

        public override int GetHashCode()
        {
            return modifier ^ key ^ hWnd.ToInt32();
        }

        public bool Register()
        {
            return RegisterHotKey(hWnd, id, modifier, key);
        }

        public bool Unregiser()
        {
            return UnregisterHotKey(hWnd, id);
        }
    }


    public static class Constants
    {
        //modifiers
        public const int NOMOD = 0x0000;
        public const int ALT = 0x0001;
        public const int CTRL = 0x0002;
        public const int SHIFT = 0x0004;
        public const int WIN = 0x0008;

        //windows message id for hotkey
        public const int WM_HOTKEY_MSG_ID = 0x0312;
    }
}
```

Appendix E

Test case	Pass or fail	Comments
1. Enable recoding by clicking the recode button and spell a word like router. Then stop the recoding by clicking the stop button and verify whether the spelled word was converted to text correctly or not.	Pass	No
2. Enable recoding by clicking the recode button and spell a word like router. Then stop the recoding by clicking the stop button and verify whether the spelled word was recoded in the intermediate wav file without background noise or not.	Pass	This was verified using math lab code attached in the testing chapter and Resuls were attached as graphs in that chapter.
3. Enable recoding by clicking the recode button and spell a word like router. Then stop the recoding by clicking the stop button. After that click do button by selecting search as the action. Verify whether it shows the correct cluster or not.	Pass	No
4. Enable recoding by clicking the recode button and spell a word like router. Then stop the recoding by clicking the stop button. After that click do button by selecting search as the action. Verify whether it shows searched results in the web browser or not.	Pass	For this test case locally hosted wamp web server was used with dummy web sites.
5. Spell two words and check whether the second word is predicted correctly based on the first word.	Pass	Initially meta data loaded to markov model and spelled words from them.