Explanation of C# code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using NLP;
using NLP.NGrams;
using NLP.TextClassification;
using System.Text.RegularExpressions;
using System.Data.Common;

namespace AutocompleteApplication
{
    4 references
    public partial class MainForm : Form
    {
        private const string TEXT_FILE_FILTER = "Text files (*.txt)|*.txt";

        private Dictionary<string, int> unigramCounts = null;
        private Dictionary<string, int> bigramCounts = null;
        private Dictionary<string, int> trigramCounts = null;
        private List<NGram> allNGrams = null;
        private String scrapedText = "";

        int totalUnigrams = 0;
        int totalBigrams = 0;
        int totalTrigrams = 0;

        1 reference
        public MainForm()
        {
            InitializeComponent();

            // Attach event handler to the sentenceBox
            sentenceTextBox.TextChanged += SentenceTextBox_TextChanged;
            sentenceTextBox.KeyDown += SentenceTextBox_KeyDown;
        }
```

In the figure above, TEXT_FILE_FILTER is used to filter out txt files. A dictionary<string, int> is created for unigrams, bigrams and trigrams to count the number of each appearance of specific n-grams. A List<NGram> allNGrams is also created for storing all the NGrams.
A String scrapedText is created to store the text from the txt file with > 5million tokens.
The totalUnigrams, totalBigrams and totalTrigrams are all set to 0.
In MainForm(), there are event handlers for when there is text entered into the sentenceTextBox and for when keyboard keys are pressed (main function used only for tab key here).

```
private void LoadDataSet()
{
    using (OpenFileDialog openFileDialog = new OpenFileDialog())
    {
        openFileDialog.Filter = TEXT_FILE_FILTER;
        if (openFileDialog.ShowDialog() == DialogResult.OK)
        {
            scrapedText = File.ReadAllText(openFileDialog.FileName).ToLower(); // Convert to lower case
            string fileName = Path.GetFileName(openFileDialog.FileName); // File name without the file path.
            nGramsListBox.Items.Add("Loaded data file \"" + fileName);
        }
    }
}

3 references
public List<NGram> GenerateNGrams(List<string> tokens, int n, Dictionary<string, int> frequencyCounts)
{
    List<NGram> nGrams = new List<NGram>();
    if (tokens.Count < n)
    {
        return nGrams;
    }

    for (int i = 0; i < tokens.Count - n + 1; i++)
    {
        string nGramIdentifier = string.Join(" ", tokens.Skip(i).Take(n));

        // Create and store the NGram object
        NGram nGram = new NGram(nGramIdentifier);
        nGrams.Add(nGram);

        // Update frequency counts
        if (frequencyCounts.ContainsKey(nGramIdentifier))
        {
            frequencyCounts[nGramIdentifier]++;
        }
        else
        {
            frequencyCounts[nGramIdentifier] = 1;
        }
    }
    return nGrams;
}

1 reference
private void loadDataSetToolStripMenuItem_Click(object sender, EventArgs e)
{
    LoadDataSet();
    if (scrapedText != null) { tokenizeButton.Enabled = true; }
    loadDataSetToolStripMenuItem.Enabled = false;
}

1 reference
private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
    Application.Exit();
}
```

In LoadDataSet(), the function filters files with txt extensions and when a txt file is selected, it is read and assigned to the string scrapedText. The application displays that the data file has been loaded.

In GenerateNGrams(List<string> tokens, int n, Dictionary<string, int> frequencyCounts) function, the tokens are used to generate frequencyCounts of each type of n-gram and then returns all the n-grams of that type via List<NGram> nGrams. Each new n-gram of that type for each token is added to nGrams. If the frequencyCounts dictionary does not contain the n-gram, the count will be set to 1. If is already contains that n-gram, the count of that n gram will be incremented.

In loadDataSetToolStripMenuItem_Click function, the LoadDataSet() function is called and if scrapedText is not null, the tokenizeButton is enabled. The loadDataSetToolStripMenuItem is then disabled.

```
1 reference
private void tokenizeButton_Click(object sender, EventArgs e)
{
    // Split scrapedText into sentences
    var sentences = Regex.Split(scrapedText, @"(?<!\w\.\w.)(?<![A-Z][a-z]\.)(?<=\.|\?)\s*");

    // Initialize n-grams dictionaries and lists
    unigramCounts = new Dictionary<string, int>();
    bigramCounts = new Dictionary<string, int>();
    trigramCounts = new Dictionary<string, int>();
    allNGrams = new List<NGram>();

    // Process each sentence separately
    foreach (var sentence in sentences)
    {
        //Tokenize the sentence into words
        List<string> sentenceTokens = Regex.Split(sentence, @"\W+").Where(t => !string.IsNullOrWhiteSpace(t)).ToList();

        //Generate n-grams for the sentence (unigrams, bigrams, trigrams)
        if (sentenceTokens.Count > 0)
        {
            totalUnigrams += sentenceTokens.Count;
            allNGrams.AddRange(GenerateNGrams(sentenceTokens, 1, unigramCounts));

            totalBigrams += sentenceTokens.Count - 1;
            allNGrams.AddRange(GenerateNGrams(sentenceTokens, 2, bigramCounts));

            totalTrigrams += sentenceTokens.Count - 2;
            allNGrams.AddRange(GenerateNGrams(sentenceTokens, 3, trigramCounts));
        }
    }

    nGramsListBox.Items.Add("");
    nGramsListBox.Items.Add("Number of tokens: " + totalUnigrams);
    int totalNGrams = allNGrams.Count;
    int currentNGram = 0;
    //nGramsListBox.Items.Add("");

    // Compute frequency per million instances and store in NGram objects
    foreach (var nGram in allNGrams)
    {
        currentNGram++;
        double progress = 100.0 * currentNGram / (double)totalNGrams;
        string identifier = nGram.Identifier;
        //nGramsListBox.Items.Add("Processing n-gram " + currentNGram + " of " + totalNGrams + " (" + progress.ToString("F2") + "%)");

        if (identifier != null)
        {
            if (unigramCounts.ContainsKey(identifier))
            {
                nGram.FrequencyPerMillionInstances = 1000000 * unigramCounts[identifier] / totalUnigrams;
            }
            else if (bigramCounts.ContainsKey(identifier))
            {
                nGram.FrequencyPerMillionInstances = 1000000 * bigramCounts[identifier] / totalBigrams;
            }
            else if (trigramCounts.ContainsKey(identifier))
            {
                nGram.FrequencyPerMillionInstances = 1000000 * trigramCounts[identifier] / totalTrigrams;
            }
        }
    }

    nGramsListBox.Items.Add("");
    nGramsListBox.Items.Add("Generated " + totalUnigrams.ToString() +
        " Unigrams, " + totalBigrams.ToString() + " Bigrams, " + totalTrigrams.ToString() + " Trigrams.");

    tokenizeButton.Enabled = false;
    sentenceTextBox.Enabled = true;
}
```

The tokenizeButton_Click function uses Regex to split the scrapedText into sentences. The dictionaries unigramCounts, bigramCounts and trigramCounts are initialised. the allNGrams list is also initialised.
Iterating through each sentence, the tokens of each sentence are determined and stored in List<string> sentenceTokens. The totalUnigrams, totalBigrams and totalTrigrams are incremented based on the sentenceTokens. For unigrams, bigrams and trigrams, the GenerateNGrams function is called such that the return value can be added to allNGrams.
The totalUnigrams is used to print the number of tokens and check that it is more than 5 million.
Iterating through all the nGram in allNGram, the FrequencyPerMillionInstances of a specific n-gram is determined and assigned to their respective nGram objects.
The number of unigrams, bigrams and trigrams are then displayed.
The tokenizeButton is disabled and the sentenceTextBox is enabled.

```
1 reference
private List<string> PredictNextWord(string text)
{
    var predictions = new List<string>();

    // Try to predict using trigrams
    var trigramPredictions = PredictWithTrigrams(text);
    if (trigramPredictions.Count > 0)
    {
        predictions.AddRange(trigramPredictions);
    }

    // If no trigram prediction, try bigrams
    if (predictions.Count == 0)
    {
        var bigramPredictions = PredictWithBigrams(text);
        predictions.AddRange(bigramPredictions);
    }

    // If no bigram prediction, try unigrams (not for this assignment)
    /*if (predictions.Count == 0)
    {
        var unigramPredictions = PredictWithUnigrams(text);
        predictions.AddRange(unigramPredictions);
    }*/
    return predictions.Distinct().Take(3).ToList();
}

1 reference
private List<string> PredictWithTrigrams(string text)
{
    var tokens = text.Split(' ');

    if (tokens.Length < 1)
    {
        return new List<string>();
    }
    if (tokens.Length == 1)
    {
        // Use full stop followed by the word
        return allNGrams
            .Where(n => n.TokenList.Count > 2 && n.TokenList[0] == "." && n.TokenList[1] == tokens[0])
            .OrderByDescending(n => n.FrequencyPerMillionInstances)
            .Take(3)
            .Select(n => n.TokenList.LastOrDefault())
            .ToList();
    }
    else
    {
        var lastTwoWords = tokens.Skip(tokens.Length - 2).ToArray();

        return allNGrams
            .Where(n => n.TokenList.Count > 2 && n.TokenList[0] == lastTwoWords[0] && n.TokenList[1] == lastTwoWords[1])
            .OrderByDescending(n => n.FrequencyPerMillionInstances)
            .Take(3)
            .Select(n => n.TokenList.LastOrDefault())
            .ToList();
    }
}
```

The List<string> PredictNextWord(string text) function predicts the next word by calling the PredictWithTrigrams and PredictWithBigrams functions. The prediction of each is added to the var predictions then three distinct predictions are obtained

The List<string> PredictWithTrigrams(string text) function first splits the text into individual tokens. If there are no tokens, then an empty string is returned. If there is one token, then the first token is “.” while the second token is the one token. Using this combination, and the words with highest FrequencyPerMillionInstances, the next word can be predicted. 3 of such words are obtained and returned. If there is more than one token, the last two tokens in the text are used

along with FrequencyPerMillionInstances to predict the next word. Again, three of such words are obtained and returned.

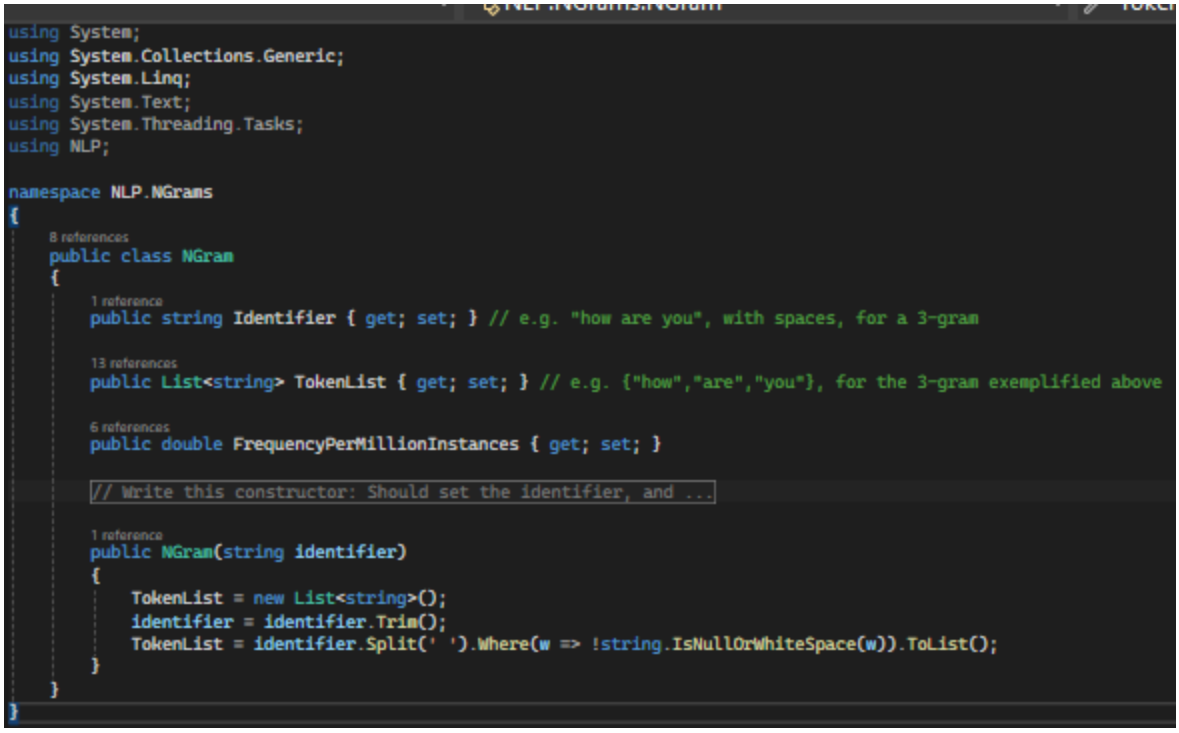
```
1 reference
private List<string> PredictWithBigrams(string text)
{
    var tokens = text.Split(' ');
    if (tokens.Length < 1)
    {
        return new List<string>();
    }
    var lastWord = tokens.Last();

    return allNGrams
        .Where(n => n.TokenList.Count > 1 && n.TokenList[0] == lastWord)
        .OrderByDescending(n => n.FrequencyPerMillionInstances)
        .Take(3)
        .Select(n => n.TokenList[1])
        .ToList();
}

/*
private List<string> PredictWithUnigrams(string text)
{
    return allNGrams
        .OrderByDescending(n => n.FrequencyPerMillionInstances)
        .Take(3)
        .Select(n => n.TokenList[0])
        .ToList();
}*/

1 reference
private void SentenceTextBox_TextChanged(object sender, EventArgs e)
{
    string inputText = sentenceTextBox.Text;

    //Only predict when the last character is a space
    if (!string.IsNullOrEmpty(inputText) && inputText.EndsWith(" "))
    {
        string trimmedInputText = inputText.Trim();
        //Get the predicted word
        List<string> predictions = PredictNextWord(trimmedInputText);

        nGramsListBox.Items.Clear();
        foreach (var prediction in predictions)
        {
            nGramsListBox.Items.Add(prediction);
        }
        //Ensures sentenceTextBox gets focus
        sentenceTextBox.Focus();
    }
}
```

In List<string> PredictWithBigrams(string text), the text is split into tokens. If there are no tokens, then an empty string is returned. If not, the last token is obtained and used to predict the next word based on words with the highest FrequencyPerMillionInstances. 3 of such words are obtained and returned.

In the void SentenceTextBox_TextChanged() function, the text in the sentenceTextBox is obtained. The text is checked that it ends with “ “ and not empty or null. After which, it is trimmed and used to predict the next word by calling PredictNextWord function. The predictions obtained are then printed out on nGramsListBox. The focus is then shifted to sentenceTextBox

```
1 reference
private void SentenceTextBox_KeyDown(object sender, KeyEventArgs e)
{

    //nGramsListBox.Items.Add("Entered keydown funtion"); //Debugging

    //Ensures sentenceTextBox gets focus
    sentenceTextBox.Focus();

    if (e.KeyCode == Keys.Tab)
    {
        //nGramsListBox.Items.Add("Tab registered"); //Debugging

        e.SuppressKeyPress = true; //Prevent default tab behaviour,which is, being entered into text

        //Ensures sentenceTextBox gets focus
        sentenceTextBox.Focus();

        string inputText = sentenceTextBox.Text;

        if (nGramsListBox.Items.Count > 0)
        {
            //nGramsListBox.Items.Add("n-grams list box items detected"); //Debugging

            //Ensures sentenceTextBox gets focus
            sentenceTextBox.Focus();

            //Append the predicted word to the input text
            string topPrediction = nGramsListBox.Items[0].ToString();
            sentenceTextBox.Text = inputText.TrimEnd() + " " + topPrediction + " ";

            //Move the cursor to the end of the text
            sentenceTextBox.SelectionStart = sentenceTextBox.Text.Length;

            //Ensures sentenceTextBox retains focus
            sentenceTextBox.Focus();
        }
    }
}
```

In the SentenceTextBox_KeyDown function, the key entered is checked if it is a Tab. If it is, then the default tab behaviour is suppressed. The focus is shifted to sentenceTextBox. Using the top predicted word displayed in nGramsListBox, the text in the sentenceTextBox is updated with it.



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

namespace NLP.NGrams
{
    8 references
    public class NGram
    {
        1 reference
        public string Identifier { get; set; } // e.g. "how are you", with spaces, for a 3-gram

        13 references
        public List<string> TokenList { get; set; } // e.g. {"how","are","you"}, for the 3-gram exemplified above

        6 references
        public double FrequencyPerMillionInstances { get; set; }

        // Write this constructor: Should set the identifier, and ...

        1 reference
        public NGram(string identifier)
        {
            TokenList = new List<string>();
            identifier = identifier.Trim();
            TokenList = identifier.Split(' ').Where(w => !string.IsNullOrWhiteSpace(w)).ToList();
        }
    }
}
```

In the NGram class constructor, the TokenList is initialised, the identifier is set to identifier.Trim() to remove trailing spaces. Then the TokenList is set with the identifier being split into individual tokens and converted to a list.