PLAGIARISM DETECTION USING SEMANTIC ANALYSIS

Khalid Shams

Student ID: 02201081

School of Engineering and Computer Science

April 2010

BRAC University, Dhaka, Bangladesh

DECLARATION

I hereby declare that this thesis is based on the results found by myself. Materials of work found by other researcher are mentioned by reference. This thesis, neither in whole nor in part, has been previously submitted for any degree.

Signature of Supervisor

Signature of Author

ACKNOWLEDGMENTS

Special thanks to Dr. Md. Khalilur Rhaman for accepting the difficult task of supervising this work to completion and giving me time out of his busy schedules to consider this work.

Abstract

Plagiarism in the sense of “theft of intellectual property” has been around for as long as humans have produced work of art and research. However, easy access to the Web, large databases, and telecommunication in general, has turned plagiarism into a serious problem for publishers, researchers and educational institutions. Plagiarism detection is a technique to find out the theft of scientific paper, literary works, source code etc.

An existing method to find out similar documents is to use Self-Organizing Maps (SOMs)[1]. But there are some efficiency challenges like processing time arise in creating these maps. To facilitate recognition of plagiarism, Researchers[2,3] at MIT used a set of low-level syntactic structures to evaluate content and expression in a document. However, we think only syntactic structures may not give optimal output in detecting plagiarism because it may not always detect the insight meaning.

To detect plagiarism, our idea is to propose a synonym and antonym based framework to evaluate text similarity with respect to the similarity of content between the original and plagiarized document. Rather using low-level syntactic structures i.e. Context-free Grammar (CFG)[4], synonymic features of sentences which we think will improve the overall combat against plagiarism.

TABLE OF CONTENTS

Page

Introduction

Easy access to the Web, large databases, and telecommunication in general, has turned plagiarism into a serious problem for publishers, researchers and educational institutions. A survey (released in June, 2005) conducted as part of Center of Academic Integrity's Assessment project reveals that 40% of students admitted to engaging in plagiarism as compared to 10% reported in 1999. Plagiarism now is not limited to just cut and paste; synonymising and translation technologies are giving a new dimension to plagiarism. Increased rate of plagiarism hurts quality of education received by students; facilitating detection of plagiarism can help teachers control this damage.

To prevent plagiarism, in the recent years many commercial and academic products have been developed i.e. Turnitin®[5] Mydropbox®[6]. Most of these products identify verbatim plagiarism. But yields poor result against paraphrased documents. Another method is to use prominent keyword searching technique to evaluate text similarity to assessment of content similarity and use features such as bag of words to find similar/relevant documents. But keywords are not sufficient for capturing expression similarity and thus need a more accurate representation of text for this task. There is a promising research done by the researchers of MIT at their CSAIL. They give more emphasize on semantic characteristic. They evaluate text similarity with respect to similarity of content and expression. But the weakness is that they used Context-Free Grammar (CFG) to represent these ideas.

Though they have used some semantic characteristics which we think can be enriched by using a synonym and antonym based framework. The reason why we propose this framework is that in practical situation people alter word by using synonym and antonym when they plagiarize intentionally. Thus it can analyze semantic features of the sentences. Because of these characteristics we hope that it will be possible to capture the meaning of words in sentences to compare the original and plagiarize document and will yield better result over CFG.

2. Related study

2.1 Definition of Plagiarism:

Plagiarism is defined as “Plagiarism is the incorporation of someone else's work without providing adequate credit[7].” Plagiarism is not always intentional or stealing some things from someone else; it can be unintentional or accidental and may comprise of self stealing.
There are many definitions of what constitutes plagiarism, and we will look at some of them in more detail below.

According to the Merriam-Webster Online Dictionary[8], to "plagiarize" means:

- to steal and pass off (the ideas or words of another) as one's own
- to use (another's production) without crediting the source
- to commit literary theft
- To present as new and original an idea or product derived from an existing source.

In other words, plagiarism is an act of fraud. It involves both stealing someone else's work and lying about it afterward. Plagiarism relates to the theft or borrowing of published work without the proper attribution or acknowledgement of source. We define plagiarism as the use of material (text, pictures, movies, etc.) without the exact specification of source; be it in unchanged form or in some kind of derivative.

Although Plagiarism and IPR violations are not a new phenomenon, the new media, particularly the internet is effectively taking it to far greater heights. Beyond print media, infringements can now occur in all types of digitized forms, including volatile forms such as SMS, Chat and Mail. The web has brought about an environment for ‘rapid generation of publications’ mainly through the instantaneous access to numerous sources of information.

Plagiarism is a major concern, particularly in an academic environment, where it could affect both the credibility of institutions as well as its ability to ensure quality of its student. Plagiarism

has been constantly on the rise, largely attributed to the Internet and web. Many students tend to take plagiarism lightly and consider a varying degree of copying to be acceptable.

2.2 Plagiarism Detection:

Plagiarism detection presents many problems in itself, as plagiarism does not always contain an obvious copying of paragraphs. There are situations where plagiarism may involve the copying of smaller chunks of content, which could further be transformed effectively to make it extremely difficult to detect. It is also possible that copied text can be translated into another language. One also has to be aware that plagiarized documents may also not always be available in digital form. There are also situations where a document is available in a digital form but it is not accessible by the detection system.

So, Plagiarism detection can in no way be considered a proof beyond doubt. It is simply employed to indicate that plagiarism may have occurred. As such, if suspicion arises based on the findings of a plagiarism detection system, a manual check is always necessary to verify this. Entirely automated plagiarism detection will result in false positives, which could be terrible.

2.3 Existing tools or methods to detect plagiarism:

Usual Approach

The usual approach for detecting plagiarism splits a document into a (large) set of 'fingerprints'[9]. A set of fingerprint contains pieces of text that may overlaps with one another. A fingerprint is then used as a query to search the web or a database, in order to estimate the degree of plagiarism. Most currently available software packages employ this technique. Variations between packages are only in the fingerprints used and search engines employed. The advantage of this method is that it is fairly stable against the re-arranging of text. It is however not stable against synonyms and translations.

Stylometry

Stylometry is an attempt to analyze writing styles based on text similarity patterns. A particular text can be compared with the typical writing style of an individual based on his or her past works. Alternatively the text in a single paragraph can be compared with the overall style of writing as found throughout a paper. As opposed to the other methods mentioned, Stylometry is able detect plagiarism without the need for an external corpus of documents. It can be applied to detect essential patterns within documents that capture style parameters that include syntactic forms, text structure as well as the usage of key terms. The detection of plagiarism within the document domain or without any external reference is well described as "intrinsic plagiarism detection[10]" by Eissen and Stein [Eissen & Stein 2006].

Integrating Search Application Programmers Interface (API)

A home-grown plagiarism detection method built on top of Google's search API has surprisingly produced superior results as compared to leading software packages in the industry such as Turnitin® and Mydropbox®. This is mainly due to Google's indexing of many more web sites as compared to these plagiarism detection tools. The limitation of employing Google's free API, has however restricted their system to 1000 queries a day. As Google does not license their Search engine, they maintain the exclusive control to numerous potential applications. This will also increase the reliance of publishers on these global search engines.

Things which are related to understand Context-free grammar (CFG)

Word categories: Traditional parts of speech

Noun: Names of things	boy, cat, truth
Verb: Action or state	become, hit
Pronoun: Used for noun	I, you, we
Adverb: Modifies V, Adj, Adv	sadly, very
Adjective: Modifies noun	happy, clever

Conjunction: Joins things	and, but, while
Preposition: Relation of N	to, from, into
Interjection: An outcry	ouch, oh, alas, psst

Constituency

The idea: Groups of words may behave as a single unit or phrase i.e. sentences have parts, some of which appear to have subparts. These groupings of words that go together are called constituents. Constituents are usually named as phrases (**Constituent Phrases**) based on the word that heads the constituent:

E.g. Noun Phrase

[Kermit the frog], [They], [December twenty-sixth], [The reason he is running for president] etc.

Kermit the frog comes on stage.

They come to Massachusetts every summer.

December twenty-sixth comes after Christmas.

The reason he is running for president comes out only now.

Other phrases:

[The man from Amherst] is a Noun Phrase (NP) because the head man is a noun.

[Extremely clever] is an Adjective Phrase (AP) because the head clever is an adjective.

[Down the river] is a Prepositional Phrase (PP) because the head down is a preposition.

[Killed the rabbit] is a Verb Phrase (VP) because the head killed is a verb.

The constituent can be placed in a number of different locations.

Example:

Constituent = Prepositional phrase: On December twenty-sixth.

On December twenty-sixth I'd like to fly to Florida.

But cannot be used be by split apart.

*[On December] I'd like to fly [twenty-sixth] to Florida.

Context-free grammar

Context-free grammar is the most common way of modeling constituency.

CFG = Context-Free Grammar = Phrase Structure Grammar = BNF = Backus-Naur Form.

The idea of basing a grammar on constituent structure dates back to Wilhem Wundt (1890), but not formalized until Chomsky (1956), and, independently, by Backus (1959).

Context-free grammar

G = T, N, S, R

• T is set of terminals (lexicon)

• N is set of non-terminals For NLP, we usually distinguish out a set

P ⊂ N of preterminals which always rewrite as terminals.

• S is start symbol (one of the nonterminals)

• R is rules/productions of the form X , where X is a nonterminal and is a sequence of terminals and nonterminals (may be empty).

• A grammar G generates a language L.

An example context-free grammar

$G = \langle T, N, S, R \rangle$

$T = \{that, this, a, the, man, book, flight, meal, include, read, does\}$

$N = \{$S, NP, NOM, VP, Det, Noun, Verb, Aux$\}$

$S = S$

$R = \{$

S → NP VP	Det → *that* | *this* | *a* | *the*
S → Aux NP VP	Noun → *book* | *flight* | *meal* | *man*
S → VP	Verb → *book* | *include* | *read*
NP → Det NOM	Aux → *does*
NOM → Noun	
NOM → Noun NOM	
VP → Verb	
VP → Verb NP	

$\}$

Application of grammar rewrite rules

S → NP VP	Det → *that* | *this* | *a* | *the*
S → Aux NP VP	Noun → *book* | *flight* | *meal* | *man*
S → VP	Verb → *book* | *include* | *read*
NP → Det NOM	Aux → *does*
NOM → Noun	
NOM → Noun NOM	
VP → Verb	
VP → Verb NP	

S → NP VP
→ Det NOM VP
→ *The* NOM VP
→ *The* Noun VP
→ *The man* VP
→ *The man* Verb NP
→ *The man read* NP
→ *The man read* Det NOM
→ *The man read this* NOM
→ *The man read this* Noun
→ *The man read this book*

Grammaticality a CFG defines a formal language = the set of all sentences (strings of words) that can be derived by the grammar. Sentences in this set said to be grammatical. Sentences outside this set said to be ungrammatical.

Parsing context-free grammars

We want to run the grammar backwards to find the structure. Parsing can be viewed as a search problem. We search through the legal rewritings of the grammar. We want to find all structures matching an input string of words. The Cocke-Younger-Kasami (CYK) algorithm solves the second of these problems, using a table data-structure called the char. This basic technique can be extended (e.g. Earley's algorithm) to handle grammars that are not in Chomsky Normal Form and to linear-time parsing for special types of CFGs.

3. Similarity detection

3.1 Similarity detection or pattern recognition by using vector space model to determine cosine[9]

A popular approach to similarity detection or pattern recognition is the use of a vector space model to determine cosine (i.e. angular) similarity among vectors of keywords/function-words extracted from the text under inspection.

Suppose we have a set of English text documents and wish to determine which document is most relevant to the query "the brown cow." A simple way to start out is by eliminating documents that do not contain all three words "the," "brown," and "cow," but this still leaves many documents. To further distinguish them, we might count the number of times each term occurs in each document and sum them all together; the number of times a term occurs in a document is called its term frequency. However, because the term "the" is so common, this will tend to incorrectly emphasize documents which happen to use the word "the" more, without giving enough weight to the more meaningful terms "brown" and "cow". Also the term "the" is not a good keyword to distinguish relevant and non-relevant documents and terms like "brown" and "cow" that occur rarely are good keywords to distinguish relevant documents from the non-relevant documents. Hence an inverse document frequency factor is incorporated which diminishes the weight of terms that occur very frequently in the collection and increases the weight of terms that occur rarely.

To elaborate more let us take an example of two sentences.

Text A: “A rainy day with a cold wind”

Text B: “A sunny day with blue sky”

Each text is represented in a word frequency table as follows:

Text A:	Text B:	Complete vocabulary:
a: 2	a: 1	a
rainy: 1	blue: 1	blue
day: 1	day: 1	cold
with: 1	sunny: 1	day
cold: 1	sky: 1	rainy
wind: 1	with: 1	sky
		sunny
		wind
		with

The representation of the two pieces of text as vectors based against the vocabulary is: Text A= {2,0,1,1,1,0,0,1,1} and Text B= {1,1,0,1,0,1,1,0,1}. Now let us take some text for similarity detection e.g. C: “A cold day”. The vector representation is C= {1,0,1,1,0,0,0,0,0}.

$$\frac{\text{Vector-A} \bullet \text{Vector-C}}{|\text{Vector-A}||\text{Vector-C}|}$$

The cosine similarity measure between text A and C is calculated using formula

Calculations give us similarity measure of 0.769 between document A and C and 0.471 between B and C. Thus one can make assumption of similarity even if the two pieces of text are not completely identical.

Advantage

Ranked retrieval: Uses a ranked retrieval approach - the system responds to a search query by ranking all documents in the corpus based on its estimate of their relevance to the query.

Terms: Terms used on searching are weighted by importance.

Partial matches: This analysis can catch the partial matches.

Accuracy:

Accuracy is good if the domain is limited and fixed. But it gives a low performance for large corpus of data.

Drawback

Assumes terms are independent. Weighting is intuitive, but not very formal. The plagiarists today are becoming aware of limitations of existing systems and avoid detection by using linguistic tools as demonstrated in one example above. They can replace functional words (keywords) after small intervals by using synonyms, retaining the idea or concept behind the sentences, yet remain undetected.

3.2 Using Syntactic Information to Identify Plagiarism

How the job is done?

This research experiment identifies classes for different syntactic expressions for the same content, called “syntactic elements of expression”. These elements of expression include: different variations of initial and final phrases of a sentence, argument structures of verb phrases and syntactic classes of verb phrases. All possible variations are considered to combat initial and final phrase structure alterations.

The order of phrases in a sentence can shift the emphasis of a sentence, can attract attention to particular pieces of information and can be used as an expressive tool.

1.

(a) Martha can finally put some money in the bank.

(b) Martha can put some money in the bank, finally.

(c) Finally, Martha can put some money in the bank.

2.

(a) Martha put some money in the bank on Friday.

(b) On Friday, Martha put some money in the bank.

(c) Some money is what Martha put in the bank on Friday.

(d) In the bank is where Martha put some money on Friday.

So, all the possibilities are considered here.

This research experiment also enriches its syntactic elements of expressions by employing Levin’s classes [Levin 1993] of verbs. In Levin’s classes verbs are classified using various syntactic alterations a verb is subject to, and the classes of verbs with

similar meanings. These features are combined to create further elements of expression for testing data (including English translations of literary work by different translators). This data is then used for recognition of paraphrased writings with similar contents.

1. Base Form

- Nora sent the book to Peter.
- NP + V + NP + PP.

2. Dative Alternation

- Nora sent Peter the book.
- NP + V + NP + NP.

Advantage

This approach can recognize titles even when they are paraphrased and the same features help improve identification of pairs of chapters that are paraphrases of each other, despite the content these chapters share with the rest.

Accuracy:

Syntactic elements of expression improve the performance of tf-idf(term frequency–inverse document frequency)-weighted keywords in recognizing pairs of paraphrased chapters significantly in terms of precision, recall, and measure. The results presented show a significantly better average of similarity detection over baseline/conventional approaches.

Drawback

This is a computationally expensive approach compared to conventional content recognition approaches such as comparing tf-idf weighted keywords, function words, distribution of word lengths and sentence lengths.

3.3 Latent semantic Indexing (LSI)

Latent semantic Indexing (LSI) or Latent semantic Analysis[10,11,12] is a technique in natural language processing, in particular in vectorial semantics, of analyzing relationships between a set of documents and the terms they contain by producing a set of concepts related to the documents and terms. A matrix of words and related segments is used to build a word to concept semantic domain space. The text needed to be checked for similarity with this domain space is also represented in document vector form. This kind of system which detects semantic similarities to grade some writing can also be used effectively for paraphrased plagiarism detection.

Advantage

This concept space typically can be used to:

- Compare the documents in the concept space (data clustering, document classification).
- Find similar documents across languages, after analyzing a base set of translated documents (cross language retrieval).
- Find relations between terms (synonymy and polysemy).

Accuracy: In the case of plagiarism detection we are usually dealing with a very large corpus of textual information making such analysis not as yet practical. Even with a singular value decomposition approach in LSA to reduce word and context matrix, the matrix dimensions are still large and the vector space analysis is computationally demanding.

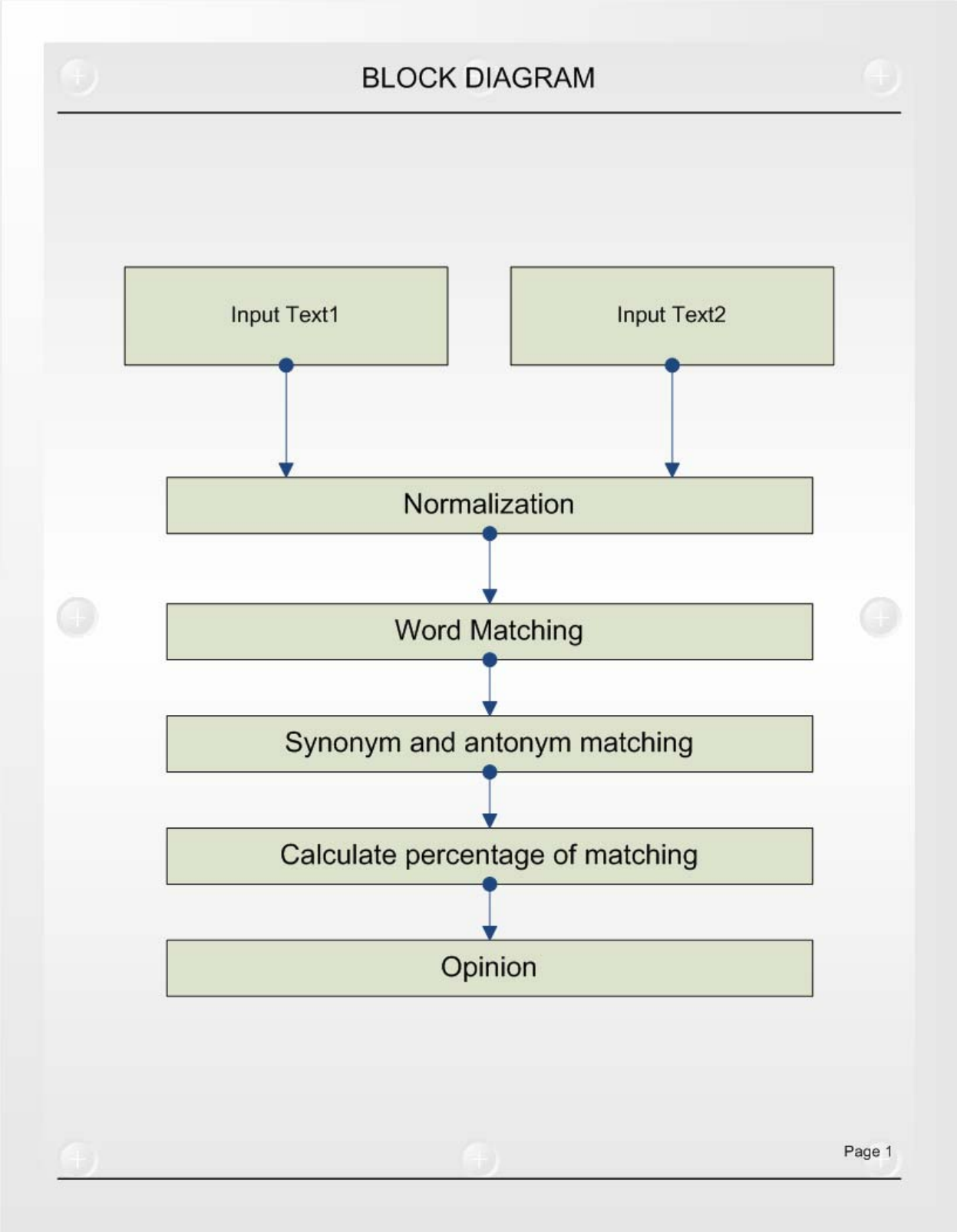
Drawback: The resulting dimensions of the matrix might be difficult to interpret. LSA cannot capture polysemy (i.e., multiple meanings of a word). Each occurrence of a word is treated as having the same meaning due to the word being represented as a single point in space.

4. Model

4.1 Proposed Model

Due to the complexities and certain drawbacks of existing plagiarism detection system, we are proposing a model to detect plagiarism by fetching word by word comparison on first place regardless of grammars. Then for the mismatched words we will check synonym and antonym similarity between the mismatched words.

4.1 Architecture



4.2 Proposed algorithm

Algorithms used in this system are described according to the block diagram.

On the “Normalization” phase we split all the sentences into string. Then we omit all the prepositions, articles, punctuations, auxiliary verbs from these strings.

After that we move on to the “Word Matching” phase. Here each string or word has been checked with string from text input box two. A potential match is counted as weight of one (1).

On the “Synonym and antonym matching” phase we look up for synonyms and antonyms of the mismatched words from input box one and try to match them with the mismatched words of input text box two. Here we weighted the synonyms according to S. I. Hayakawa in his book “The Penguin modern guide to synonyms and related words”[13].

Then on the “Calculate percentage of matching” phase we made a ratio between the two documents over the similarity and number of words. Based on that the program will give an opinion whether the document should be checked for plagiarism or not.

4.3 Database

We have created a demo database where all the words i.e. synonyms and antonym are kept. All the words are indexed. We have taken these sample words from thefreedictionary.com. they are kept in a 1-D array named “WordList”. Then we have created a 2-D array named “Map” to store the entire synonym and antonym information based on the index of the “WordList” array.

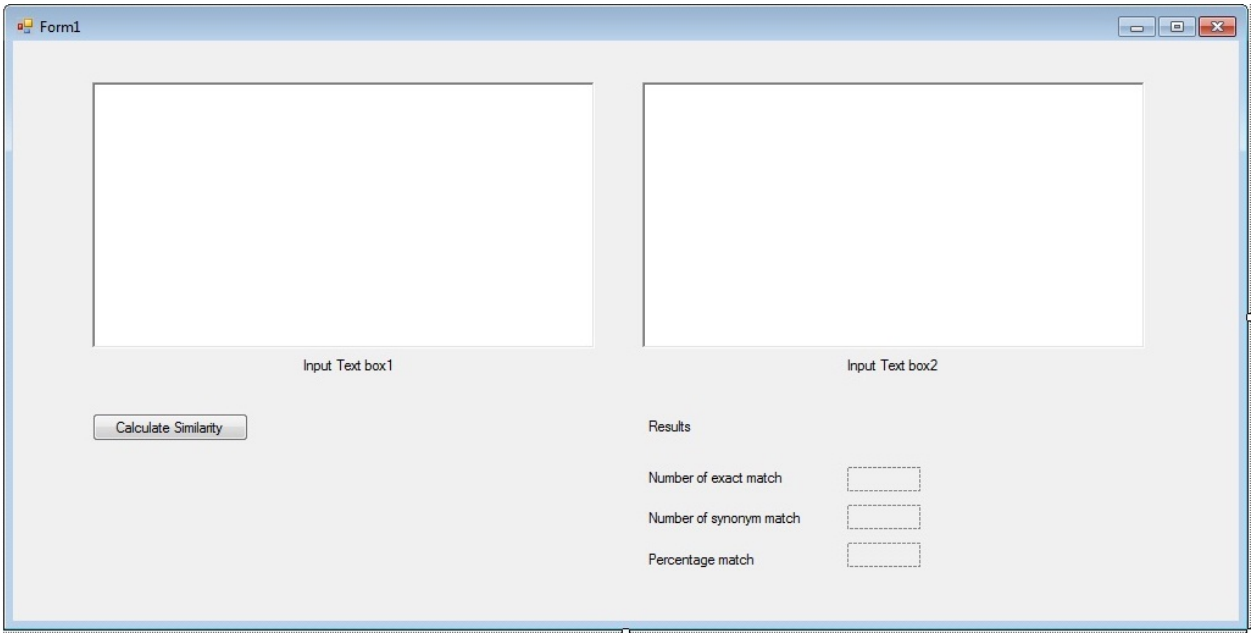
Word List

	1	**compare**	analyze, examine
	2	**monozygotic**	identical, same
	3	**twin**	pair, similitude
	4	**dizygotic**	dizygous, nonidentical
	5	**researcher**	investigator, scientist
	6	**begin**	start
	7	**separate**	divide, differentiate
	8	**effect**	result, cause
	9	**gene**	cistron, factor
	10	**environment**	surroundings, condition
	11	**analyze**	compare, examine
	12	**examine**	analyze, compare
	13	**identical**	same, monozygotic
	14	**pair**	twin, similitude
	15	**similitude**	twin, pair
	16	**dizygous**	dizygotic, nonidentical
	17	**investigator**	researcher, scientist
	18	**start**	begin
	19	**divide**	separate, differentiate
	20	**result**	effect, cause
	21	**cause**	result, effect
	22	**condition**	environment, surroundings
	23	**nonidentical**	dizygous,dizygotic
	24	**scientist**	investigator, researcher
	25	**differentiate**	separate, divide
	26	**cistron**	gene, factor
	27	**factor**	gene,cistron
	28	**same**	identical, monozygotic

Number of synonyms	2
Index of the first synonym	11
Index of the second synonym	12
Weight of the first synonym	0.6
Weight of the first synonym	0.7

Map structure of the word “compare

4.4 Proposed Interfaces



Proposed Interface 1

Here is the proposed interface that will be created C# .net platform 3.5.

5. **Analysis**

We have not test our program on a lot of data. As a result we cannot tell the accuracy of our program by this time. According to our hypothesis we hope that it will yield better result than the other plagiarism detection system.

We are going to release a prototype very soon including words from the WordNet[14] dictionary from Princeton University.

Thus it will enhance our accuracy level.

6. **Problem discussion**

First limitation of the system is that we have not used a grammatical structure to analyze the documents. In this hypothesis we are using the semantic meaning of words by measuring the weight of the synonyms and antonyms.

Few lexicographers argue that there cannot be a synonym for any word because every word is different by its phonetic, origin and uses. This argument has not been considered as in real life plagiarism is done widely by using synonym and antonym.

7. Summary:

In this report we have proposed an idea to detect plagiarism using synonym and antonym similarities. We think our proposed system will work much efficiently and effectively over SOMs. Because this new method will definitely reduce the time needed to analyze documents.

Our proposed idea is at a very initial stage. It might change its direction time to time. What we hope is that our proposed method will produce comparatively better result to detect plagiarism in near future.

8. **References**

[1] Paola Merlo, James Henderson, Gerold Schneider and Eric Wehrli 2003. Learning Document Similarity Using Natural Language Processing

[1] Ozlem Uzuner, Boris Katz and Thade Nahnsen 2005. Using Syntactic Information to Identify Plagiarism

[1] Ozlem Uzuner, Randall Davis, Boris Katz, Using Empirical Methods For Evaluate Expression and Content Similarity, Proceedings of the 2nd Workshop on Building Educational Applications Using NLP,June 2005

[4] http://www.cs.rochester.edu/~nelson/courses/csc_173/grammars/cfg.html

[5] **turnitin**.com/

[6] www.**mydropbox**.com

[1] Hermann Maurer, Frank Kappe and Bilal Zaka, Plagiarism - A Survey, Journal of Universal Computer Science, 2006

[8] www.**merriam**-**webster**.com

[9] Spärck Jones, Karen - A statistical interpretation of term specificity and its application in retrieval, (1972).

[10] Dumais, S. T., Furnas, G. W., Landauer, T. K. and Deerwester, S. (1988), "Using latent semantic analysis to improve information retrieval." In Proceedings of CHI'88: Conference on Human Factors in Computing, New York: ACM, 281-285.

[11] Deerwester, S., Dumais, S. T., Landauer, T. K., Furnas, G. W. and Harshman, R.A. (1990) "Indexing by latent semantic analysis." Journal of the Society for Information Science, 41(6), 391-407.

[12] Foltz, P. W. (1990) “Using Latent Semantic Indexing for Information Filtering". In R. B. Allen (Ed.) Proceedings of the Conference on Office Information Systems, Cambridge, MA, 40-47.

[13] Samuel Ichiyé Hayakawa, P. J. Fletcher , The Penguin modern guide to synonyms and related words, 1987

[14] http://wordnetweb.princeton.edu/perl/webwn

Index

CODE

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

namespace ThsImp
{
    class Program
    {
        static void Main(string[] args)
        {
            //TextReader tr = new StreamReader("wordlist.txt");


            string wordListReader = File.ReadAllText("wordlist.txt");
            string[] wordlist = wordListReader.Split(new char[] { ' ', '\t',
'\r', '\n' }, StringSplitOptions.RemoveEmptyEntries);

            //for (int a = 0; a < 28; a++)
            //    wordlist[a] = tr.ReadLine();

            //tr.Close();

            for (int a = 0; a < wordlist.Length; a++)
                Console.WriteLine(a + " " + wordlist[a]);

            int[,] map = new int[5, 28];


            string fileContent = File.ReadAllText("index.txt");

            string[] integerStrings = fileContent.Split(new char[] { ' ',
'\t', '\r', '\n' }, StringSplitOptions.RemoveEmptyEntries);

            int[] integers = new int[integerStrings.Length];

            for (int n = 0; n < integerStrings.Length; n++)
                integers[n] = int.Parse(integerStrings[n]);

            int count = 0;

            for (int col = 0; col < 28; col++)
                for (int row = 0; row < 5; row++)
                {
                    map[row, col] = integers[count];
                    count++;
                }
```

```
            //for (int c = 0; c < 28; c++)
            //    for (int r = 0; r < 5; r++)
            //        Console.WriteLine(c + " " + wordlist[c] + " List " +
map[r, c]);


            string line1, line2;

            // Read the file and display it line by line.

            StreamReader file1 = new StreamReader("sample1.txt");
            StreamReader file2 = new StreamReader("sample2.txt");

            line1 = file1.ReadLine();
            line2 = file2.ReadLine();

            //Console.WriteLine(line1);
            //Console.WriteLine(line2);

            file1.Close();
            file2.Close();

            string line1Lower = line1.ToLower();
            string line2Lower = line2.ToLower();

            string[] temp = { "by", "to", "can", "of", "the", "from" };
            int index;

            for (int x = 0; x < temp.Length; x++)
            {
                while (line1Lower.IndexOf(temp[x]) != -1)
                {
                    index = line1Lower.IndexOf(temp[x]);
                    line1Lower = line1Lower.Remove(index, temp[x].Length +
1);

                }
                while (line2Lower.IndexOf(temp[x]) != -1)
                {
                    index = line2Lower.IndexOf(temp[x]);
                    line2Lower = line2Lower.Remove(index, temp[x].Length +
1);

                }
            }

            //Console.WriteLine(line1Lower);
            //Console.WriteLine(line2Lower);

            line1Lower = line1Lower.Replace(",", " ");
            line2Lower = line2Lower.Replace(",", " ");

            char[] seperators = { ' ', ',', '.' };
```

```
            string[] l1 = line1Lower.Split(seperators);
            string[] l2 = line2Lower.Split(seperators);

            for (int a = 0; a < l1.Length; a++)
                Console.WriteLine(a + " " + l1[a]);

            for (int a = 0; a < l2.Length; a++)
                Console.WriteLine(a + " " + l2[a]);

            //search words

            int[] index_l1 = new int[l1.Length];
            int[] index_l2 = new int[l2.Length];

            for (int i = 0; i < index_l1.Length; i++)
                index_l1[i] = -1;

            for (int i = 0; i < index_l2.Length; i++)
                index_l2[i] = -1;

            int counter1 = 0;
            int counter2 = 0;

            int sum = 0;


            for (int i = 0; i < l1.Length; i++)
                for (int j = 0; j < l2.Length; j++)
                {

                    if (l1[i] == l2[j])
                    {
                        if (counter1 < index_l1.Length && counter2 <
index_l2.Length)
                        {
                            index_l1[counter1++] = i;
                            index_l2[counter2++] = j;
                        }
                        sum = sum + 1;
                        Console.WriteLine("i = " + i + " j = " + j);
                        break;
                    }

                }

            for (int i = 0; i < index_l1.Length; i++)
                Console.WriteLine("index = " + i + " value = " +
index_l1[i]);



            //for (int i = 0; i < index_l2.Length; i++)
            //    Console.WriteLine("index = " + i + " value = " +
index_l2[i]);
```

```
            //for (int i = 0; i < l1.Length; i++)
            //    for (int j = 0; j < l2.Length; j++)
            //        ;

            Console.WriteLine("Sum = " + sum);

            //search

            int index_key = 0;
            int match = 0;
            double fraction = 0;

            for (int iterator1 = 0; iterator1 < index_l1.Length; iterator1++)

                if (index_l1[iterator1] == -1)
                {
                    Console.WriteLine("iterator1 = " + iterator1);
                    Console.WriteLine(l1[iterator1]);
                    index_key = Array.IndexOf(wordlist, l1[iterator1]);
                    int synsize = map[0, index_key];
                    Console.WriteLine("synsize = " +synsize+ "word = " +
wordlist[index_key] + " syn1 = " + wordlist[map[1, index_key]] + " syn2 = " +
wordlist[map[2, index_key]]);

                    if (synsize == 2)
                    {
                        int m1 = map[1, index_key];
                        int m2 = map[2, index_key];

                        match = Array.IndexOf(l2, wordlist[m1]);

                        if (match < 0)
                            match = Array.IndexOf(l2, wordlist[m2]);

                        if (match > 0)
                            fraction = Convert.ToDouble(sum + map[4,
index_key] / 10);

                        Console.WriteLine("match = " + match);

                    }
                    else
                    {
                        int syn1 = map[1, index_key];
                        match = Array.IndexOf(l2, wordlist[syn1]);

                        if (match > 0)
                            fraction = Convert.ToDouble(sum + map[3,
index_key] / 10);

                    }


                }
```

```
            Console.WriteLine(sum);
            Console.WriteLine(l2.Length);
            fraction = (Convert.ToDouble(sum) /Convert.ToDouble( l2.Length))
* 100.0;

            Console.WriteLine("Average match = " + fraction + " %");


            Console.Read();

        }
    }
}
```
