HÁSKÓLINN Í REYKJAVÍK
REYKJAVIK UNIVERSITY

Reykjavík University

- Computer Science -

Natural Language Processing

Final Project Report

Fall Semester 2013/2014

Christopher Dörge (Kennitala 080888-3929)

David Müller (Kennitala 130389-3769)

1 Introduction

Intrinsic plagiarism detection is a very active field of research and deals with identifying portions of text in a document, that might originate from another author. The approach of intrinsic plagiarism detection is based on the assumption that every author has a different writing style. Utilizing stylometric features like the average sentence length, the frequency of certain characteristic words or the use of punctuation in a text can be used to reveal stylistic variations in a suspicious document [7].

In contrast to intrinsic plagiarism detection being limited to the source document itself, extrinsic plagiarism detection relies on a reference corpus. This means, that other documents or even web search is utilized for identifying passages partly derived from other documents [8]. Common methods for extrinsic plagiarism detection involve n-gram based comparisons of the suspicious text and potential source texts or purely string-based approaches, as the computation of the longest common subsequence of two texts [2].

In this report, we experiment with an intrinsic method using the vector space model. The aim is to analyze the impact of the source document language on the overall detection results. Therefore we developed a program to detect plagiarism intrinsically, which uses the same algorithm to classify suspicious parts inside of English and German documents.

2 Vector Space Models and Related Work

Several methods exist in the area of intrinsic plagiarism detection – see [9] and [4]. One popular method is the Vector Space Model, where several characteristic features are combined together into a single vector representing a portion of the text. The first challenge is finding an appropriate sized chunk of the text [3] for further analysis. If the selected chunk size is too large, shorter plagiarized passages may not stand out. On the other hand, if the chunk size is too short a lot of false positives may be generated.

Having selected a sensible chunk size, the second challenge consists of selecting appropriate features to include in the feature vector for every chunk. Thereafter, all of the individual feature vectors can be compared to the document's overall mean feature vector, for the purpose of revealing outliers [1]. The main advantage of the Vector Space Model is the flexibility of its approach, i.e. it is not limited to certain features. In theory, every possible analysis method could be included in the vector calculation. The disadvantage of this flexibility is, on the other hand, that not every additional feature is beneficial to the overall detection rate.

In previous related work, research concerning Vector Space Models has been carried out on the impact of using several different features on the overall detection performance [3]. Furthermore, the appropriate size of the selected chunk / sliding window has been analyzed in [1]. We are not aware of previous work in which the impact of the source language on intrinsic plagiarism detection performance is examined.

3 Detecting Outliers

Every dimension inside the target vector for a single unit is represented by a certain feature. A unit can be a sentence or a paragraph, depending on the setting for the unit to evaluate.

To be able to compare those unit vectors, the mean vector of the document is calculated by adding the vectors of all units and dividing by the total count of units. The cosine similarity measure is then used to calculate the similarity of two vectors, i.e. the cosine of the angle between a unit vector A and a mean vector B:

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

Having calculated all individual vectors and their (cosine) similarity to the documents mean vector, outliers can be detected, for example using standard deviation coefficient [6]:

$$cos(A, B) < B - \epsilon * stddev$$

A chunk is considered to be plagiarized, if the cosine similarity between the analyzed vector and the documents mean vector is smaller than the documents mean minus the product of the standard deviation of all chunks and a (to be determined) optimal constant epsilon.

4 Vector Components

The detection performance of an implementation utilizing the Vector Space Model strongly depends on the individual features. If meaningless features are selected for the vector and given high weights, the detection performance will decline. Thus, it is imperative to use the most important features when constructing the vector and assign sensible weights to them according to their importance.

We adapt and extend the approach described by Zechner et al. [1] with an augmented feature set partly derived from Grieve [5] and Meyer zu Eissen and Stein [7]. Our feature set consists of:

(a) Average word frequency class

For any document or unit, it is possible to rank words based on their total occurrence. This ranking can be used to group words within frequency classes. The frequency class of a word is defined by:

$$class(w) = \lfloor \log(occ(w*)/occ(w)) \rfloor$$

where *occ(w*)* is the absolute number of occurrences of the most frequent word within a document and *occ(w)* is the number of occurrences of a specific word in the same text [7]. Each calculated word frequency class becomes one component of the feature vector.

The value of this component is the amount of words belonging to this class within the analyzed unit.

(b) Dictionary Analysis

The general usage of foreign words or the number of mistakes in a text can reveal much about an author's fingerprint. Wordlists are used to determine whether a given token is part of a dictionary or not. Therefore, we add a feature which stands for the count of words found in the wordlist inside the analyzed unit.

(c) Parts-Of-Speech Analysis

We use the Stanford Tagger [10] to tokenize the text and assign a part-of-speech (PoS) tag to every token. Each possible tag in the underlying tagset is represented by a dimension in the vector. The value of this component is the total amount of occurrences of words with the given PoS tag inside the analyzed unit.

(d) Punctuation Analysis

The amount of punctuation tokens (e.g. comma, dot, colon, hyphen) that are found in one analyzed chunk also receives a dimension in the vector.

(e) Word Frequency Analysis

The word frequency from a collection of commonly used words containing both stop words (of, and, or, …) and pronouns (I, me, he, her, myself …) is included in the vector as well. Zechner et al. [1] include the frequency of every word from the given collection of stop words and pronouns in the vector as a single component.

(f) Average Word Length, Sentence Length

An important characteristic of an author's personal writing style is the preferred sentence length and the average word length within one chunk. This holds true especially for languages like German, where infinitely long words can be produced by compounding.

(g) Higher Level Language Features (Tenses, Active/Passive)

Authors tend to have a personal preference for a tense form, which they use commonly. The same holds true for the use of the active/passive form.

(h) Chunking

The selection of a sensible chunk size is an important part of intrinsic plagiarism detection [1]. Several approaches can be applied:

1) Fixed Amount of Characters
Using a fixed character count to chop the source document into chunks, for example 2000 characters. This ensures an equal amount of text for every chunk, but is very likely to intersect sentences. One downside of intersecting sentences is the confusion of the part-of-speech information, because certain tokens typically appear mostly at certain positions in a sentence.

2) Fixed Amount of Sentences
Using a fixed amount of sentences for every chunk, for example one sentence. This ensures coherent sentences, but could cause problems because the length of the chunks are most likely different.

3) Using Paragraphs
Using whole paragraphs of the text as chunks. This approach is based on the assumption that plagiarism typically does not occur on a sentence level, but rather in semantic units like paragraphs. This comes with the same downside as in (2), because paragraphs are going to vary in length. A shorter paragraph will then stand out and may have a higher chance of producing a false positive.

5 Sliding Window Approach

To improve the detection, combining the previously described approaches (1), (2) or (3) with a sliding window can be beneficial. For (1), the sliding window could ensure that only whole sentences are used as units. For (3), k sentences around the current sentence unit are taken into every analyzed chunk.

Our detection algorithm operates with various sliding window sizes starting from 4 units and ending at 7 units. The algorithm includes 4 units to the left and 4 units to

the right of the current sentence, if 4 is our sliding window size. The following example shows this process with a sliding window size of 2 and using sentences as units:

Parameters: Sliding Window = 2, Units = Sentences

Source Text: Sentence 1. Sentence 2. Sentence 3. Sentence 4. Sentence 5. Sentence 6. Sentence 7. Sentence 8.

Unit 1: Sentence 1. Sentence 2. Sentence 3. Sentence 4. Sentence 5.

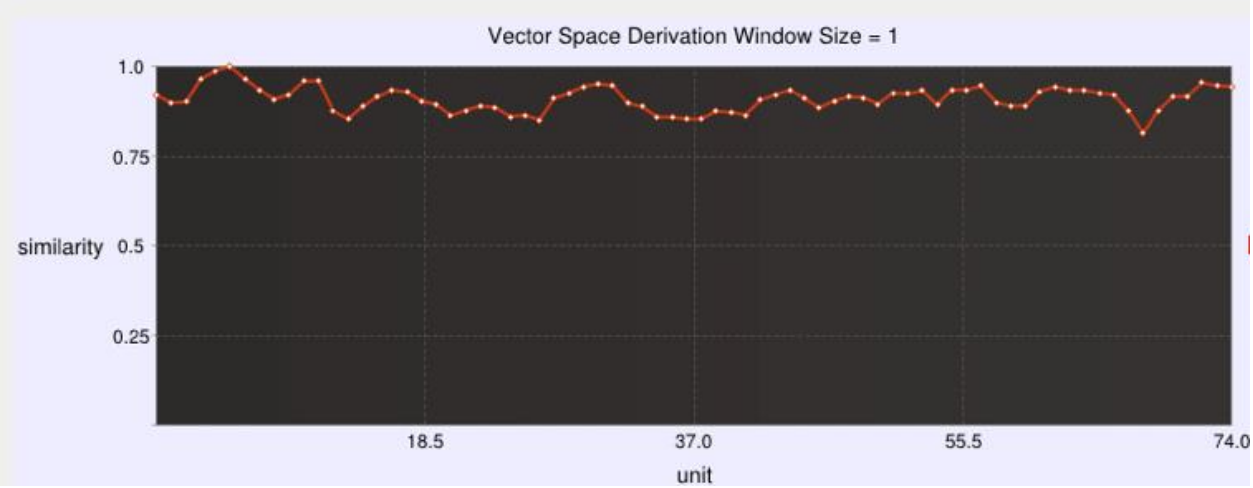
Unit 2: Sentence 2. Sentence 3. Sentence 4. Sentence 5. Sentence 6.

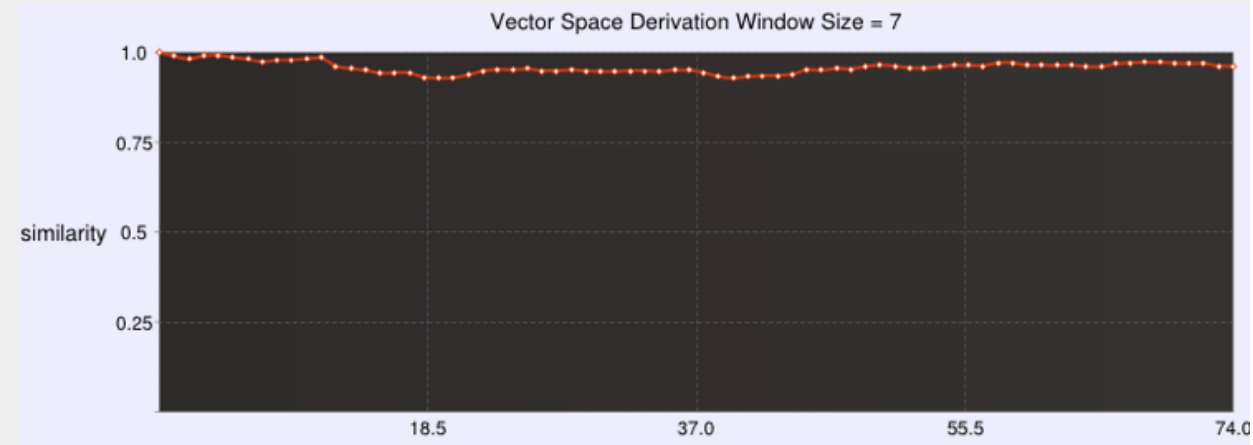
Unit 3: Sentence 3. Sentence 4. Sentence 5. Sentence 6. Sentence 7.

. . .

The improvement of using a sliding window approach is the flattening of outliers. If a sentence in a document is very short, it will most likely be detected as plagiarism right away. By using a sliding window, those outliers are flattened to a certain extend. If 2 or more sentences in a row are plagiarized, the sliding window will more likely identify them as outliers.

The flattening process is shown in the following 2 images:





When using varying sliding window sizes, different units will be detected as outliers.

For example, a sliding window size of 4 might identify units **<3, 9, 14>** as plagiarized, a sliding window of 5 will do so for units **<3, 9, 15>** and a sliding window of 6 will identify **<3, 8, 14>** as plagiarized.

This is the point, where the parameter **Sliding Window Percentage** becomes important. In our example above, we receive the following detection coverages:

- **Unit 3**: 100%
- **Unit 8**: 33%
- **Unit 9**: 66%
- **Unit 14**: 66%
- **Unit 15**: 33%

If we have a **Sliding Window Percentage** of 1.0, only outliers will be included that are detected by every sliding window size. In our example, only unit 3 will be reported as outlier. With a **Sliding Window Percentage** of 0.1, every outlier will be reported that occurs in at least 10% of all sliding windows.

6 Corpus Generation

We originally planned to use the PAN Corpus[1], which was recommended to us by a research member of the International Competition on Plagiarism Detection[2]. Sadly, this corpus did not contain German-German plagiarism, meaning German texts with inserted parts of other German documents. As our main research aim is to analyze the language dependence in intrinsic plagiarism detection, we definitely need to have corpora with English-English plagiarism and with German-German plagiarism. Thus, we had to create an own corpus.

6.1. Crawler

Our main source for German documents is "Project Gutenberg"[3]. This portal contains lots of German books from popular literature, which are available for free (Example: "Der Prozeß" by Franz Kafka[4]). We crawl several pages from Projekt Gutenberg randomly by varying the book id and the page number. This is possible, because the URL's of the books at Projekt Gutenberg are always structured in the same fashion:

`http://gutenberg.spiegel.de/buch/<book id>/<page of book>`

Every page contains approximately 2500 words.

The primary source for English documents is "Classic Reader"[5], which is comparable to Projekt Gutenberg, as it also contains popular literature of many different authors. For example, Bleak House by Charles Dickens is available at this URL:

`http://www.classicreader.com/book/221/1/`

[1] http://www.uni-weimar.de/en/media/chairs/webis/research/corpora/
[2] http://pan.webis.de/
[3] http://gutenberg.spiegel.de/
[4] http://gutenberg.spiegel.de/buch/157/2
[5] http://www.classicreader.com/book/221/2/

6.2. Methodology

Plagiarized documents are created by randomly inserting one or more sentences (up to a whole paragraph) at a random position into the source document. Important aspects of this process are:

1) The source document has to have a minimum length to be included into the corpus – otherwise, determining the individual writing style of an author will not be possible. In our test runs, 1500 words turned out to be a good number.
2) The length of the inserted plagiarized sentence / sentences also needs to exceed a certain minimum. We found out, that 200 characters seem to be a good number. This number appears to be realistic, considering that a single short sentence with e.g. 5 words will most likely never be plagiarized exclusively.
3) The overall amount of inserted plagiarism can not exceed a certain percentage of the source documents length. We selected a lower limit of 2% and an upper limit of 10% as plagiarism insertion margin.

The generated plagiarized documents are saved in simple text files which can be easily supplied as input documents to our intrinsic plagiarism detector.

6.3. Saving the Position of the Plagiarized Parts

After generating the corpus, the position of the plagiarized parts contained in the documents needs to be kept. We decided to use a JSON file to store these positions.

An example of the structure is presented here:

```
4   "0002.txt": {
5     "plagiarizedSentences": [
6       33,
7       34,
8       55,
9       56
10    ],
11    "plagiarizedParagraphs": [
12      6,
13      9
14    ]
15  }
```

Here, the document “0002.txt” contains 4 plagiarized sentences. For the purpose of analyzing the effect of using whole paragraphs as units compared to single sentences, the paragraph numbers in which the plagiarism was inserted are also saved.

7 Performing the Evaluation

7.1. Evaluation Parameters

In in order to be able to analyze the whole generated corpus in one run, our aim was to automatize the evaluation to a point, where no manual interaction of us is necessary any more.

To achieve this, the program needs to vary a lot of input parameters, that all change the detection results. Those input parameters are briefly explained in the next section.

7.1.1 Unit Definition

To be able to identify semantic units as “plagiarism block” and in order to compare the detection performance of a sentence based approach to the paragraph based approach, we made our algorithm capable of dealing with paragraphs as units. This was done because plagiarism is very likely to happen on paragraph level instead of sentence level.

To conclude, this parameter triggers if sentences or paragraphs are used as units (see section “Vector Components”).

7.1.2 Word Level Normalization

The ordinary method in Vector Space Model based Intrinsic Plagiarism Detection algorithms just counts all occurrences of a certain feature regardless of the sentence length. If a sentence contains 4 pronouns, the number 4 is registered for the PoS-tag “Pronoun” in this unit.

We added an optional normalization mechanism, which is triggered by this parameter. When Word Level Normalization is activated, we divide the counted feature number by the words in the unit.

This way, one pronoun in a unit with 10 words yields the same value (1/10) as a unit with 4 pronouns and 40 words length (4/40).

7.1.3 Epsilon

A unit is considered to be plagiarized, if the cosine similarity between the analyzed vector and the documents mean vector is smaller than the documents mean minus the product of the standard deviation of all unit and the constant epsilon.

$$cos(A, B) < B - \epsilon * stddev$$

This constant influences the detection dramatically, because it determines the threshold of a unit being reported as plagiarized. As suggested by Halvani [6], epsilon should be a to determined "small constant".

7.1.4 Feature Weights

Each of the vector space model features (Average Word Frequency, Average Word Length, Dictionary Analysis, POS Analysis, Punctuation Frequency, Sentence Length, Stop Word Frequency, Active/Passive, Tense) can be assigned a weight to balance its importance in the detection process.

If we assign a weight of 0 to one of the features, it will not be considered at all. On the other hand, assigning a weight higher than the weight of the other features balances the detection towards this feature.

The challenge here is identifying the best balance of weights for all the features.

7.2. Evaluation Setup

To estimate the quality of the achieved results we had to evaluate the outcome of our different runs. Precision and recall are based on an understanding and a measure of relevance. To calculate those two values, we had to find the TP (true positives), FP (false positives), FN (false negatives) and TN (true negatives) by checking every unit if it was reported as outlier and if this allocation was correct.

At first, we set all the relevant previously described parameters for our evaluation run and start by iterating over every document in the corpus.

The evaluation itself is performed within *evaluateDocument().* After that, the plagiarized units of the documents are then received from the JSON document created by the corpus generator.

The following pseudo algorithm illustrates the setup used for performing an evaluation run:

```
SetEvaluationParameters( ... );

for each document in corpus
{
    var detectedOutliers = evaluateDocument()
    var realOutliers = parseJsonDocumentWithRealOutliers(document)

    for each unit in document
    {
        if (unit is in detectedOutliers)
        {
            if (unit is in realOutliers)
                truePositive++
            else
                falsePositive++
        }
        else
        {
            if (unit is in realOutliers)
                falseNegative++
            else
                trueNegative++
        }
    }
}

calculatePrecision(truePositive, falsePositive);
calculateRecall(truePositive, falseNegative);
calculateAccuracy(truePositive, falsePositive, trueNegative, falseNegative);
```

Precision in the topic of plagiarism means the fraction of achieved suspicious units that are plagiarized and is calculated as follows:

$$precision = \frac{TP}{TP + FP}$$

Recall in our case means the fraction of achieved plagiarized units that are successfully retrieved and is calculated as follows:

$$recall = \frac{TP}{TP + FN}$$

Initially we wanted to evaluate the accuracy of our results, which is the percentage of units that are classified correctly.

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

By looking at the results of our first evaluation runs, we determined that the usage of accuracy as quality indicator is not feasible. Since our training and test documents only contain few plagiarized sentences, but much more non-plagiarized sentences, the accuracy was high in such cases when our detection algorithm could not find any or less suspicious parts.

Therefore we decided to discard the utilization of the traditional accuracy measure and use the F1-measure instead, which represents the harmonic mean of precision and recall and computes better results in situations where the minority class is more important:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

The results of our evaluation runs are saved in a big CSV file to be able to analyze the reported results later on.

7.3. Varying the Parameters

One big "side challenge" is identifying the best combination of parameters to achieve a good overall detection result. Varying all the input parameters by hand individually is a very cumbersome task, so we set up an architecture consisting of several nested for-loops. Those for-loops iterate over every sensible parameter range to identify the best combination.

```
//are we using paragraphs
for (int useParagraphs = 0; useParagraphs <= 1; useParagraphs++)
//divide factor by the word number
for (int wordLevel = 0; wordLevel <= 1; wordLevel++)
//factor for cosine similiarity
for (double epsilon = 0.5; epsilon <= 1.0; epsilon += 0.5)
//percentage of sliding window occurences, until we treat it as a plagiate
for (double outliersOccurrence = 0.1; outliersOccurrence <= 1.0; outliersOccurrence += 0.45)

//4 entries in feature vector
for (double AverageWordFrequency = 12; AverageWordFrequency <= 25; AverageWordFrequency += 12)
//1 entry in feature vector
for (double AverageWordLength = 50; AverageWordLength <= 100; AverageWordLength += 50)
//1 entry in feature vector
for (double Dictionary = 50; Dictionary <= 100; Dictionary += 50)
//(num of different pos) entries in map -> 13
for (double Pos = 4; Pos <= 8; Pos += 4)
//1 entry in feature vector
for (double Punctuation = 50; Punctuation <= 100; Punctuation += 50)
//1 entry in feature vector
for (double SentenceLength = 50; SentenceLength <= 100; SentenceLength += 50)
//473 entries for every stop word
for (double StopWord = 0.12; StopWord <= 0.23; StopWord += 0.11)
//1 active/passive entry in the feature vector
for (int activePassive = 0.5; activePassive <= 1.0; activePassive += 0.5)
//1 tense entry in the feature vector
for (double tense = 0.5; tense <= 1.0; tense += 0.5)
{
    SetParameters( ... );
    (Precision, Recall, F1) = EvaluateAllDocumentsOfCorpus ( ... );
    SaveResultsAndParamsToCSV( ... );
}
```

As shown, we use 13 nested for loops to iterate over sensible ranges for all available input parameters. In the setup which is shown here, we perform

```
2 * 2 * 3 * 3 * (2 ^ 9) = 18932
```

iterations to evaluate all possible combinations. In every iteration, the whole corpus is evaluated and the Recall, Precision and F1-measure are saved alongside with the used parameter combination.

The sensible parameter ranges of the loops were determined by the number of entries in the feature vector. We have - for example - 4 *Average Word Frequencies* entries in the feature vector. To ensure a comparable influence of every analysis method on the overall feature vector, we set the lower parameter boundary to 50% and the upper boundary to 100%. This means that for our example *Average Word Frequencies* with 4 entries, the lower boundary is $50 / 4 \sim 12$ and the upper boundary is $100 / 4 \sim 25$. By

applying this selective weighting mechanism, we guarantee to put equitable emphasis on every feature. Otherwise, the evaluation would be heavily biased towards analysis methods with multiple feature vector entries.

7.4. Detection Improvement and Setup

On a generated corpus containing 100 documents, 80 documents were picked for training. Conclusively, 20 documents are saved for the test. This setup is the same for the English and the German corpus.

Afterwards, all the 18932 parameter combinations are evaluated on both the English and the German corpus[6].

The evaluation result on the whole "language corpus" for every parameter combination is appended to a CSV file together with the used setup. This enables us to detect the most effective parameter set afterwards:

```
1  time;truePositiveTotal;falsePositiveTotal;trueNegativeTotal;falseNegativeT
   otal;precisionTotal;recallTotal;accuracyTotal;useParagraphs;wordLevel;epsi
   lon;outliersOccurrence;AverageWordFrequencyAnalysis;AverageWordLengthAnaly
   sis;DictionaryAnalysis;PosAnalysis;PunctuationAnalysis;SentenceLengthAnaly
   sis;StopWordAnalysis
2
3  1385088949;4;165;375;19;0.023668639053254437;0.17391304347826086;0.6731793
   960923623;0;0;0.5;0.1;12.0;50.0;50.0;4.0;50.0;50.0;0.12
4
5  1385088964;4;165;375;19;0.023668639053254437;0.17391304347826086;0.6731793
   960923623;0;0;0.5;0.1;12.0;50.0;50.0;4.0;50.0;50.0;0.22999999999999998
6
7  1385088972;5;171;369;18;0.028409090909090908;0.21739130434782608;0.6642984
   014209592;0;0;0.5;0.1;12.0;50.0;50.0;4.0;50.0;100.0;0.12
8
9  1385088982;5;171;369;18;0.028409090909090908;0.21739130434782608;0.6642984
   014209592;0;0;0.5;0.1;12.0;50.0;50.0;4.0;50.0;100.0;0.22999999999999998
```

7.5. Creating the Deviation Table

Theoretically, we could have proceeded by selecting the best parameter combination and running it on the testing corpus right away. However, having all this data at hand,

[6] As a sidenote, one complete run for both corpora took 55 hours on a Quad-Core Intel i7 (4 * 3.2 GHz) with 8 gigabytes memory.

we decided to do a complete visualization of both corpus runs on top of that before continuing with the evaluation on the test corpus.

Thus, we implemented a script with the following capabilities:

1) Show the average Precision, Recall and F1-measure for all the iterations on the corpus.
2) Display the improvement (or deterioration, respectively) for every parameter. This is done by calculating the Precision, Recall and F1-measure of all the runs with a certain parameter and comparing them with the averages for the whole corpus. By doing this, we get the percental improvement or deterioration with respect to the average.

Those tables help us to evaluate, whether a single parameter has a more satisfying impact on both languages or not.

For the German corpus, we received the following results:

Precision	Recall	F1	Condition
+49.52%	+68.96%	+56.61%	useParagraphs = 0
-49.52%	-68.96%	-56.61%	useParagraphs = 1
-58.99%	-59.27%	-57.96%	wordLevel = 0
+58.99%	+59.27%	+57.96%	wordLevel = 1
+24.51%	+40.96%	+28.17%	epsilon = 0.5
-24.51%	-40.96%	-28.17%	epsilon = 1
+19.42%	+38.3%	+26.55%	outliersOccurrence = 0.1
+3.13%	-6.45%	-1.56%	outliersOccurrence = 0.55
-22.55%	-31.85%	-24.98%	outliersOccurrence = 1
+1.01%	+0.73%	+1.01%	AverageWordFrequencyAnalysis = 12
-1.01%	-0.73%	-1.01%	AverageWordFrequencyAnalysis = 24
+6.58%	+5.67%	+6.55%	AverageWordLengthAnalysis = 50
-6.58%	-5.67%	-6.55%	AverageWordLengthAnalysis = 100
+10.52%	+10.75%	+10.41%	DictionaryAnalysis = 50
-10.52%	-10.75%	-10.41%	DictionaryAnalysis = 100
+0.22%	-0.33%	-0.01%	PosAnalysis = 4
-0.22%	+0.33%	+0.01%	PosAnalysis = 8
-0.97%	-1.07%	-1.06%	PunctuationAnalysis = 50
+0.97%	+1.07%	+1.06%	PunctuationAnalysis = 100
-17.48%	-15.64%	-17.19%	SentenceLengthAnalysis = 50
+17.48%	+15.64%	+17.19%	SentenceLengthAnalysis = 100
+0.65%	+0.27%	+0.36%	StopWordAnalysis = 0.12
-0.65%	-0.27%	-0.36%	StopWordAnalysis = 0.23
-6.71%	-5.68%	-6.72%	UseActivePassive = 0
+6.71%	+5.68%	+6.72%	UseActivePassive = 1

Note, that we used only Active / Passive detection as higher-level NLP feature for German, because Tense detection was not feasible (see section “Problems” for further details).

The results for the English corpus are the following:

Precision	Recall	F1	Condition
+49.95%	+68.97%	+56.71%	useParagraphs = 0
-49.95%	-68.97%	-56.71%	useParagraphs = 1
-58.61%	-58.9%	-57.58%	wordLevel = 0
+58.61%	+58.9%	+57.58%	wordLevel = 1
+24.38%	+41.63%	+28.49%	epsilon = 0.5
-24.38%	-41.63%	-28.49%	epsilon = 1
+18.78%	+39.23%	+26.55%	outliersOccurrence = 0.1
+2.81%	-6.69%	-1.56%	outliersOccurrence = 0.55
-21.58%	-32.54%	-24.99%	outliersOccurrence = 1
-0.36%	+0.67%	-0.07%	AverageWordFrequencyAnalysis = 12
+0.36%	-0.67%	+0.07%	AverageWordFrequencyAnalysis = 24
+6.64%	+5.05%	+6.49%	AverageWordLengthAnalysis = 50
-6.64%	-5.05%	-6.49%	AverageWordLengthAnalysis = 100
+10.49%	+11.67%	+10.88%	DictionaryAnalysis = 50
-10.49%	-11.67%	-10.88%	DictionaryAnalysis = 100
-0.08%	-1.17%	-0.18%	PosAnalysis = 4
+0.08%	+1.17%	+0.18%	PosAnalysis = 8
-1.46%	-1.82%	-1.43%	PunctuationAnalysis = 50
+1.46%	+1.82%	+1.43%	PunctuationAnalysis = 100
-17.41%	-15.08%	-16.95%	SentenceLengthAnalysis = 50
+17.41%	+15.08%	+16.95%	SentenceLengthAnalysis = 100
-0.79%	-0.79%	-0.63%	StopWordAnalysis = 0.12
+0.79%	+0.79%	+0.63%	StopWordAnalysis = 0.23
+3.38%	+3.51%	+3.47%	UseTenses = 0
-3.38%	-3.51%	-3.47%	UseTenses = 1

Note, that we used only Tense detection as higher-level NLP feature for English, because Active / Passive detection was not feasible (see section “Problems” for further details).

Overall, we received higher average Precision, Recall and F1-measure values for the German corpus.

German Corpus

- Average Precision: 6.5%
- Average Recall: 41.6%
- Average F1: 9.2%

English Corpus

- Average Precision: 5.4%
- Average Recall: 29.2%
- Average F1: 6.9%

Interestingly, some parameter settings (e.g. using sentences as units instead of paragraphs) show a positive impact on both corpora. Contrary, other settings (e.g. putting higher emphasis on the average word frequency analysis) show an improvement for the German corpus but worsen the evaluation for the English corpus and vice versa.

Speaking about the "higher level" NLP features, the active/passive feature seems to deteriorate the overall detection results (for German) whereas the tense feature improves the results the evaluation on the English corpus.

Furthermore, the influence of the Stop Word Analysis for our result seems to be negligible and the Part of Speech Analysis only has a small impact. Going in greater detail here and examining the influence of certain parameter combinations could be the subject of a further elaboration and might lead to very interesting results. However, this will not be part of this report.

7.6. Identifying the Best Parameter Combination

We proceeded by identifying the highest values for precision, recall and F1-measure for both corpora.

German corpus:

Maximum precision: 25.1%
Maximum recall: 96.2%
Maximum F1: 36.2%

Run on the German corpus with best precision:

Precision: 25.1%
Recall: 52.5%
F1: 33.9%

Run on the German corpus with best recall:

Precision: 14.3%
Recall: 96.2%
F1: 24.9%

Run on the German corpus with best F1:

Precision: 23.3%
Recall: 82.5%
F1: 36.2%

English corpus:

Maximum precision: 27.8%
Maximum recall: 71.7%
Maximum F1: 36.7%

Run on the English corpus with best precision:

Precision: 27.8%
Recall: 40%
F1: 32.8%

Run on the English corpus with best recall:

Precision: 12.6%
Recall: 71.7%
F1: 21.4%

Run on the English corpus with best F1:

Precision: 26%
Recall: 62.2%
F1: 36.7%

The best recall for the German corpus is much higher than for the English corpus (96.2% vs 82.5%), whereas the best precision is almost the same (25.1% vs. 27.1%). In terms of the combined F1-measure, both languages also perform very similarly (36.2% vs. 36.7%).

Interestingly, the average F1-measure of the English corpus is lower than for the German corpus (9.2% vs. 6.9%) but the overall highest F1-measure of 36.7% can be found in the English corpus.

We now continued by identifying the parameter combination that led to the best F1-measure for both languages:

Best parameter combination for the German corpus (F1: 36.2%)

useParagraphs = 0
wordLevel = 1
epsilon = 0.5
outliersOccurence = 0.55
AverageWordFrequencyAnalysis = 24
AverageWordLengthAnalysis = 100
DictionaryAnalysis = 100
PosAnalysis = 4
PunctuationAnalysis = 50
SentenceLengthAnalysis = 50
StopWordAnalysis = 0.12
UseActivePassive = 1

Best parameter combination for the English corpus (F1: 36.7%)

useParagraphs = 0
wordLevel = 1
epsilon = 0.5
outliersOccurrence = 0.1
AverageWordFrequencyAnalysis = 24
AverageWordLengthAnalysis = 50
DictionaryAnalysis = 50
PosAnalysis = 4
PunctuationAnalysis = 50
SentenceLengthAnalysis = 100
StopWordAnalysis = 0.23
UseTenses = 0

As expected, the best combinations resemble the deviation table almost entirely.

7.7. Testing on unseen Documents / Conclusion

The determined best parameter combinations are now tested on the 20% unseen documents, to ensure we have not overfitted the classifier and actually achieved representative results.

For comparison, the values in parenthesis show the maximum value from the training corpus.

Run on the German Test corpus with the best parameter combination:

Precision: 16.5% (23.3%)
Recall: 68.1% (82.5%)
F1: 26.6% (36.2%)

Run on the English Test corpus with the best parameter combination:

Precision: 17.4% (26%)
Recall: 58.4% (62.2%)
F1: 26.8% (36.7%)

Both F1-values drop by ~10% on the test corpus compared to the training corpus. It seems like 80 documents have not been enough to train a robust parameter combination - we expected the detection performance of the test run to be closer to the training run.

Zechner et al. [1] achieve a similar precision on the PAN competition corpus (19.7%), but have a lower F1-score (22.9%) because of their lower recall (27.2%). However, comparing the results one by one is not sensible, because the PAN corpus is structured and organized in a different way to our corpus.

8 Problems

8.1 Validity of the results using the Vector Space Model

Our results show that intrinsic mechanisms using vector space models are not satisfying to be used as a general method for detecting plagiarism. This correlates with the high amount of occurrences of false positives in the evaluation. The fundamental assumption, that a plagiarized chunk inside a text will be always mapped as an outlier in a similarity chart of all chunks in comparison to the mean is not correct. Moreover, unplagiarized units vary a lot in their composition of features. Therefore. a mechanism using the standard deviation to detect suspicious chunks may misjudge and classify a not plagiarized unit as suspicious.

The sliding window approach with its ability to flat the similarity graph has the drawback to hide units, which are plagiarized. This property is perhaps negligible, because the advantage to flat outlying regular units prevails but it is definitely a point to be remarked.

8.2 Detecting the Active/Passive Voice

For the English language, the Stanford Core NLP[7] project is available. Core NLP comes as a 250MB jar-file and can be used to detect additional features of sentences

[7] http://nlp.stanford.edu/downloads/corenlp.shtml

and produce parse trees. Detecting passive sentences is relatively straight forward, because the parse tree contains a "nsubjpass" dependency, if a sentence is in passive voice[8]. However, Stanford Core NLP is known for being exceptionally slow. Tagging performance is slowed down from ~0.2s per page to 7-10s per page, because the parse tree needs to be generated for every sentence. Manually detecting the passive voice in English (in order to speed up the detection process) does not seem to be feasible for our project[9]. This is why we only included active/passive detection in our feature vector when evaluating for the German corpus.

For the German language, we can not use Stanford Core NLP, because it only comes with parse models for English. However, detecting the German Passive Voice manually is generally easier, because it is usually built by a form of "werden", an optional object followed by the "Partizip II" or the infinitive form.

8.3 Detecting the Tense

Tense detection in English is relatively simple using the following algorithm presented by the paper "N-gram-based Tense Models for Statistical Machine Translation"[10].

[8] http://nlp.stanford.edu/downloads/dependencies_manual.pdf

[9] http://writingcenter.unc.edu/handouts/passive-voice/

[10] http://nlp.suda.edu.cn/~zhxgong/paper/emnlp2012.pdf

Algorithm 1 Determine the tense of a node.

Input:
The TreeNode of one parse tree, $leafnode$;
Output:
The tense, $tense$;
1: $tense$ = "UNK"
2: Obtaining the POS tag $lpostag$ from $leafnode$;
3: Obtaining the word $lword$ from $leafnode$;
4: **if** ($lpostag$ in ["VBP", "VBZ"]) **then**
5: $tense$ = "$present$"
6: **else if** ($lpostag$ == "VBD"]) **then**
7: $tense$ = "$past$"
8: **else if** ($lpostag$ == "MD"]) **then**
9: **if** ($lword$ in ["$will$", "ll", "$shall$"]) **then**
10: $tense$ = "$future$"
11: **else if** ($lword$ in ["$would$", "$could$"]) **then**
12: $tense$ = "$past$"
13: **else**
14: $tense$ = "$present$"
15: **end if**
16: **end if**
17: **return** $tense$;

In the field of research, German Tense Detection has not been covered widely. We did not find a paper dealing with this special topic and there no pre-defined algorithms. As previously mentioned, Stanford Core NLP is only usable with the English language and thus can't help us here. Furthermore, the PoS which the Stanford Tagger supplies when tagging for the German language (using the NEGRA corpus[11]) are very general and do not help us. This is why we only included tense detection in our feature vector when evaluating for the English corpus.

[11] http://www.coli.uni-saarland.de/projects/sfb378/negra-corpus/negra-corpus.html

9 References

1. Zechner, M.; Muhr, M.; Kern, R.; Granitzer, M. 2009. External and Intrinsic Plagiarism Detection Using Vector Space Models. In *Proceedings of the SEPLN'09 Workshop on Uncovering Plagiarism, Authorship and Social Software Misuse.*
2. Chong, M. 2013. A Study on Plagiarism Detection and Plagiarism Direction Identification Using Natural Language Processing Techniques.
3. Stein, B.; Lipka, N.; Prettenhofer, P. 2011. Intrinsic Plagiarism Analysis. In *Journal Language Resources and Evaluation, Volume 45 Issue 1, March 2011, Pages 63-82.*
4. Seaward, L.; Matwin, S. 2009. Intrinsic Plagiarism Detection using Complexity Analysis. In *Proceedings of the SEPLN'09 Workshop on Uncovering Plagiarism, Authorship and Social Software Misuse.*
5. Grieve, J. 2007. Quantitative Authorship Attribution: An Evaluation of Techniques. In *Literary and Linguistic Computing, pp. 251–270.*
6. Halvani, O. 2011. Towards Intrinsic Plagiarism Detection.
7. Meyer zu Eissen, S.; Stein, B. 2006. Intrinsic Plagiarism Detection, In *Advances in Information Retrieval: Proceedings of the 28th European Conference on IR Research.*
8. Chong, M.; Specia, L.; Mitkov, R. 2010. Using Natural Language Processing for Automatic Detection of Plagiarism. In *Proceedings of the 4th International Plagiarism Conference (IPC 2010).*
9. Stamatatos, E. 2009. Intrinsic Plagiarism Detection Using Character n-gram Profiles. In: *3rd PAN Workshop. Uncovering Plagiarism, Authorship and Social Software Misuse.*
10. Toutanova, K.; Klein, D.; Manning, C.; Singer, Y. 2003. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In *Proceedings of HLT-NAACL 2003*, pp. 252-259.