Aplicação de Técnicas de PLN para Detecção de Plágio em Documentos

Renan Baisso
Yago Sorrilha

Aplicação de Técnicas de PLN para Detecção de Plágio em Documentos

- **Using Natural Language Processing for Automatic Detection of Plagiarism**
- **Apresentado em: Proceedings of the 4th International Plagiarism Conference 2010**
- **34 citações**

Using Natural Language Processing for Automatic Detection of Plagiarism

Miranda Chong   Lucia Specia   Ruslan Mitkov

Research Group in Computational Linguistics
University of Wolverhampton, UK
{Miranda.Chong | L.Specia | R.Mitkov} @wlv.ac.uk

Keywords: Natural Language Processing, Plagiarism Detection, External plagiarism, Plagiarism

ABSTRACT

Current plagiarism detection tools are mostly limited to comparisons of suspicious plagiarised texts and potential original texts at string level. In this study the aim is to improve the accuracy of plagiarism detection by incorporating Natural Language Processing (NLP) techniques into existing approaches. We propose a framework for external plagiarism detection in which a number of NLP techniques are applied to process a set of suspicious and original documents, not only to analyse strings but also the structure of the text, using resources to account for text relations. Initial results obtained with a corpus of plagiarised short paragraphs have showed that NLP techniques improve the accuracy of existing approaches.

1. INTRODUCTION

The ease of information sharing through the Internet has encouraged searching for literature online, which has resulted in many people, particularly in academic fields, to replicate other people's ideas or work without appropriate acknowledgement. While the prevention of such a problem is a very important direction to be followed especially for educational purposes, the detection of plagiarised cases is also necessary. Over the years many methodologies have been developed to perform automatic detection of plagiarism, including tools for natural language text detection such as Turnitin (iParadigms, 2010) and CopyCatch (CFL software, 2010), and tools for computer programming source code detection such as MOSS (Aiken, 1994).

The detection of plagiarism is not a new research area. Various approaches have been developed to deal with both external and intrinsic plagiarism on written texts (Lukashenko Graudina & Grundspenkis, 2007). External plagiarism detection consists in comparing suspicious plagiarised documents against potential original documents. Intrinsic plagiarism detection, on the other hand, consists in finding plagiarised passages within a document without access to potential original texts.

However, the methods used for plagiarism detection so far are mostly limited to a very superficial level, for example by comparing suspicious texts and original texts at the string level to check the amount of word overlapping across documents (Bull *et al.*, 2001; Badge & Scott, 2009). As a consequence, the accuracy of detection approaches is yet to reach a satisfactory level (Lyon Barrett & Malcolm, 2001). Although recent work (Ceska & Fox, 2009) has targeted pre-processing techniques to generalise documents by replacing words with their base forms, the techniques used are still very limited and no significant improvements were reported. Therefore plagiarism continues to be a growing challenge, affecting many areas - namely education, publishing and even business sectors.

Our aim is to investigate means to improve the accuracy of existing detection approaches by using Natural Language Processing (NLP) technologies. Better approaches to detection could also be used to establish a broader understanding of the importance of correct referencing and encourage the deployment of plagiarism prevention approaches. For this paper, we focus on the challenge of external plagiarism in monolingual text.

The rest of this paper is organised as follows: in Section 2 we present related work on plagiarism detection. In Section 3 we describe the methodology we chose and the experimental settings. In Section 4 we present the results from our experiments. In Section 5 we discuss the strengths and weaknesses we observed. Finally, we draw some conclusions in Section 6.

Dataset

- Corpus: Clough & Stevenson (2009)
- Dataset composto por pequenos trechos de texto em que alunos respondiam questões sobre textos extraídos da Wikipedia: 100 documentos (95 suspeitos de plágio e 5 originais)
- As respostas foram classificadas em 4 classes:
  - Near Copy: copy e paste do texto original
  - Light Revision: pequenas substituições do texto original
  - Heavy Revision: texto baseado no original realizando reestruturação e parafraseando
  - Non-plagiarism: baseado apenas nos conhecimentos dos alunos

Creating a corpus of plagiarised academic texts

Paul Clough
Department of Information Studies
University of Sheffield
p.d.clough@shef.ac.uk

Mark Stevenson
Department of Computer Science
University of Sheffield
m.stevenson@dcs.shef.ac.uk

Abstract

Plagiarism is a serious problem in higher education and generally acknowledged to be on the increase (McCabe, 2005). Text analysis tools have the potential to be applied to work submitted by students and assist the educator in the detection of plagiarised text. It is difficult to develop and evaluate such systems without examples of such documents. There is therefore the need for resources that contain examples of plagiarised text submitted by students. However, gathering examples of such texts presents a unique set of challenges for corpus construction.

This paper discusses current work towards the creation of a corpus of documents submitted for assessment in higher education that contain examples of simulated plagiarism. The corpus is designed to represent the types of plagiarism that are found within higher education as closely as possible. We describe the process of corpus creation and some features of the resulting resource. It is hoped that this resource will become useful for research into the problem of plagiarism detection.

1 Introduction

In recent years plagiarism (and its detection) has received much attention within the academic and commercial communities (e.g. (Hislop, 1998; Joy, 1999; Lyon et. al., 2001; Colberg and Kobourov, 2005; Eissen and Stein, 2006; Kang et. al., 2006). In academia students have used technology to fabricate texts (e.g. using pre-written texts from essay banks or paper mills, using word processors to manipulate texts and finding potential source texts using online search engines) and plagiarism is now widely acknowledged to be a significant and increasing problem for higher education institutions (Colwin and Lancaster, 2001; Zobel, 2004; McCabe, 2005).

The academic community have suggested a wide range of approaches to the detection of plagiarism, for example (White and Joy, 2004; Colberg and Kobourov, 2005), and many commercial systems are also available (Bull, 2001). However, one of the barriers preventing a comparison between these techniques is the lack of a standardised evaluation resource. Such a resource would enable a quantitative evaluation of existing techniques for plagiarism detection. Standardised evaluation resources have been very beneficial to a wide range of fields including Information

Pré-processamento dos dados

- **Tokenização**

```
def tokenizer(string):
    #https://www.w3schools.com/python/python_regex.asp#findall
    regex = r"\b[-a-zA-ZÀ-ÖØ-öø-ÿ0-9]+\b"
    return re.findall(regex, string)
```

- **Normalização**

```
def normalizer(word):
    return [ w.lower() for w in word]
```

- **Remoção de StopWords**

```
class StopWordsHandler:
    #https://stackoverflow.com/questions/6022764/python-removing-list-element-while-iterating-over-list/6024599
    #https://gist.github.com/sebleier/554280
    def isStopWord(self,word):
        stop_words = ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd",
        if word in stop_words:
            return True
        else:
            return False
    def removeStopWords(self,words):
        for i in list(words):
            if self.isStopWord(i) or i == '':
                words.remove(i)
            else:
                pass
        return words
```

Pré-processamento dos dados

- **Stemming**

```
class PorterStemmerCustom:
    #https://tartarus.org/martin/PorterStemmer/python.txt
    def isCons(self, letter):
        if letter == 'a' or letter == 'e' or letter == 'i' or letter == 'o' or letter == 'u':
            return False
        else:
            return True

    def isConsonant(self, word, i):
        letter = word[i]
        if self.isCons(letter):
            if letter == 'y' and self.isCons(word[i-1]):
                return False
            else:
                return True
        else:
            return False

    def isVowel(self, word, i):
        return not(self.isConsonant(word, i))

    # *S
    def endsWith(self, stem, letter):
        if stem.endswith(letter):
            return True
        else:
            return False

    # *v*
    def containsVowel(self, stem):
        for i in stem:
            if not self.isCons(i):
                return True
        return False
```

...

```
    def step5a(self, word):
        if word.endswith('e'):
            base = word[:-1]
            if self.getM(base) > 1:
                word = base
            elif self.getM(base) == 1 and not self.cvc(base):
                word = base
        return word

    def step5b(self, word):
        if self.getM(word) > 1 and self.doubleCons(word) and self.endsWith(word, 'l'):
            word = word[:-1]
        return word

    def stem(self, word):
        word = self.step1a(word)
        word = self.step1b(word)
        word = self.step1c(word)
        word = self.step2(word)
        word = self.step3(word)
        word = self.step4(word)
        word = self.step5a(word)
        word = self.step5b(word)
        return word
```

Pré-processamento dos dados

- **Lemmatisation**

```
def get_wordnet_pos(word):
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}

    return tag_dict.get(tag, wordnet.VERB)
```

```
def lemmatizer(words):
    lemmatizer = WordNetLemmatizer()
    lemmas = []

    for w in words:
        lemmas.append(lemmatizer.lemmatize(w, get_wordnet_pos(w)))
    return lemmas
```

Cálculo de Similaridade entre os textos

- **Distância entre vetores por cossenos**

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

Em que A e B são os vetores que armazenam, para cada documento, as frequências das palavras dos textos que desejamos verificar a similaridade. Isto é, no espaço vetorial dos texto, qual a proximidade dos vetores pela relação entre cossenos

- **Com isso, calculamos as similaridades de cada documento com os documentos originais, com o intuito de gerar *features* para aplicação de um classificador de textos**

Features

- **Calculamos as similaridades dos documentos após diferentes combinações de técnicas de PLN para representar *features* que representassem a similaridade entre os textos**

- **Features:**
    - **f1: similaridade apenas após a tokenização;**
    - **f2: similaridade após tokenização + remoção de StopWords**
    - **f3: similaridade após tokenização + remoção de StopWords + Stemming**
    - **f4: similaridade após tokenização + remoção de StopWords + segmentação de sentenças + Lemmatisation**

- **Adicionamos essas features em um *dataset* com uma variável resposta sobre plágio para cada um dos textos, com o intuito de treinar e testar um classificador Naive Bayes**

Features

	File	Category	f1	f2	f3	f4
0	g0pA_taska.txt	non	0.645986	0.439486	0.509229	0.495879
5	g0pB_taska.txt	non	0.658039	0.186013	0.536226	0.467069
10	g0pC_taska.txt	heavy	0.838993	0.727802	0.775872	0.755284
15	g0pD_taska.txt	cut	0.938483	0.876625	0.902144	0.885649
20	g0pE_taska.txt	light	0.993699	0.985442	0.988046	0.987032

Classificador de textos com Naive Bayes

- **Conjunto de dados: todos os textos produzidos pelos alunos, com as respectivas medidas de similaridade mencionadas no slide anterior e a classificação dentre as 4 possíveis (Near Copy, Light Revision, Heavy Revision, Non-plagiarism)**

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics

features = ['f1', 'f2', 'f3', 'f4']
target = ['Category']
def NBProcess(df, target, features):
    #X features que utilizaremos para treinar o modelo
    X = df[features]

    #Y variável resposta, no caso nível de plágio
    y = df[target]


    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2)

    gauss = GaussianNB()

    gauss.fit(X_train, y_train)

    y_pred = gauss.predict(X_test)

    accuracy = metrics.accuracy_score(y_test, y_pred)

    print('Matriz de confusão:')
    print(metrics.confusion_matrix(y_test, y_pred))
    print()
    print(f'Acurácia da classificação: {accuracy*100}%')
    print('Resumo da classificação:')
    print(metrics.classification_report(y_test, y_pred))
```

Resultados

- **Utilizando a metodologia descrita anteriormente, chegamos aos seguintes resultados para a matriz de confusão das classes exploradas:**

	Esperado				
	Classes	non	heavy	light	cut
Classificado	non	4	0	1	1
	heavy	2	1	0	0
	light	0	5	2	2
	cut	1	0	0	0

Tabela 1 - Matriz de Confusão

Resultados

- **Relatório de desempenho composto por precisão, recall e f1-score para cada classe predita pelo modelo de Naive Bayes utilizado**

classes	precision	recall	f1-score
non	0.80	0.89	0.84
heavy	0.50	0.60	0.55
light	0	0	0
cut	0	0	0
geral	0.31	0.35	0.29

Tabela 2 - Métricas de Desempenho

- **Utilizando 80% do conjunto de dados para treinar o algoritmo e 20% para teste, obtivemos uma acurácia de 57,89%**
- **O algoritmo desenvolvido possui complexidade O(n), sendo n o número de palavras no corpus**

Conclusões

- Obtivemos um desempenho menor com relação ao artigo que serviu como base para desenvolvimento de nossa metodologia.
  No artigo original os resultados obtidos para a acurácia ficaram entre 60 e 70%, entretanto não aplicamos todas as técnicas do artigo em nossa metodologia, justificando o valor da acurácia de 57,89%.

- Para melhoria do trabalho realizado e futuros projetos, pode-se combinar as técnicas utilizadas com os demais procedimentos de PLN mais avançados, como a modelagem com N-gramas e nomeação de entidades, para gerar melhor desempenho e melhores resultados na classificação de textos em diferentes níveis de plágio

- Além disso, para melhorar a acurácia e o desempenho do classificador de Naive Bayes pode-se utilizar um corpus com maior massa de dados classificados, dado que não foram utilizados muitos dados para treino e teste do modelo.